PaYnter Documentation

Release 0.1.4

Stefano Bertoli

Nov 28, 2019

Contents:

PaYnter is a Python Library that let you procedurally generate images with handy features that emulates what you can find in any image editing software like Photoshop, GIMP, Krita, and similars.

CHAPTER 1

Getting started

1.1 Installation

1.1.1 From Pip

The recommended way to use Paynter is to download it with pip with the command:

..code-block:: pip3 install paynter

Also, if you already installed Paynter and you want to upgrade it to the latest release you can do so by running the pip upgrade command:

..code-block:: pip3 install –upgrade paynter

1.1.2 From Source

PaYnter can also be used by downloading the source code from the GitHub repository and adding the *paynter* folder inside your project directory.

1.2 Usage

After you got Paynter inside your environment, you can use it by simply importing the library inside your main Python script through

```
#Import the paynter library
from paynter import *

#Instantiate your Paynter object
paynter = Paynter()

#Unleash your creativity here:
```

1.3 Example.py

Since getting used to the objects inside this library can be a bit tricky for coding novices, I made an Example file that covers the basics principles of Paynter.

NB: To be able to use the brushes created by example.py you also need to download the *res* folder where all the used brushtips are stored.

CHAPTER 2

Reference

2.1 The `Paynter` Module

The core module of the PaYnter project, this class will do most of the work when you use this library.

2.1.1 The `Paynter` Class

class paynter.paynter.**Paynter**

This class is the main object of the library and the one that will draw everything you ask. To create this class you can use the default constructor.

```
from paynter import *
P = Paynter()
```

Paynter.**renderImage**(*output=''*, *show=True*)

Renders the `Image` and outputs the final PNG file.

Parameters

- **output** – A string with the output file path, can be empty if you don't want to save the final image.
- **show** – A boolean telling the system to display the final image after the rendering is done.

Return type Nothing.

2.1.2 Getters and setters

Paynter.**setColor**(*color*)

Sets the current `Color` to use.

Parameters **color** – The `Color` to use.

Return type Nothing.

Paynter.**setColorAlpha**(*fixed=None*, *proportional=None*)
Change the alpha of the current Color.

Parameters

- **fixed** – Set the absolute 0-1 value of the alpha.
- **proportional** – Set the relative value of the alpha (Es: If the current alpha is 0.8, a proportional value of 0.5 will set the final value to 0.4).

Return type Nothing.

Paynter.**getColorAlpha**()
Retrieve the alpha of the current Color.

Return type A float 0-1 value of the current Color alpha.

Paynter.**getBrushSize**()
Retrieve the size of the current Brush.

Return type An integer value of the current Brush size in pixels.

Paynter.**swapColors**()
Swaps the current Color with the secondary Color.

Return type Nothing.

Paynter.**setBrush**(*b*, *resize=0*, *proportional=None*)
Sets the size of the current Brush.

Parameters

- **brush** – The Brush object to use as a brush.
- **resize** – An optional absolute value to resize the brush before using it.
- **proportional** – An optional relative float 0-1 value to resize the brush before using it.

Return type Nothing.

Paynter.**setMirrorMode**(*mirror*)
Sets the mirror mode to use in the next operation.

Parameters **mirror** – A string object with one of these values : '', 'h', 'v', 'hv'. "h" stands for horizontal mirroring, while "v" stands for vertical mirroring. "hv" sets both at the same time.

Return type Nothing.

2.1.3 Drawing functions

Paynter.**drawPoint**(*x*, *y*, *silent=True*)
Draws a point on the current Layer with the current Brush. Coordinates are relative to the original layer size WITHOUT downsampling applied.

Parameters

- **x1** – Point X coordinate.
- **y1** – Point Y coordinate.

Return type Nothing.

Paynter.**drawLine**(*x1*, *y1*, *x2*, *y2*, *silent=False*)
Draws a line on the current Layer with the current Brush. Coordinates are relative to the original layer size WITHOUT downsampling applied.

Parameters

- **x1** – Starting X coordinate.
- **y1** – Starting Y coordinate.
- **x2** – End X coordinate.
- **y2** – End Y coordinate.

Return type Nothing.

`Paynter.`**`drawPath`**(*pointList*)

Draws a series of lines on the current `Layer` with the current `Brush`. No interpolation is applied to these point and *`drawLine()`* will be used to connect all the points lineraly. Coordinates are relative to the original layer size WITHOUT downsampling applied.

Parameters **`pointList`** – A list of point like `[(0, 0), (100, 100), (100, 200)]`.

Return type Nothing.

`Paynter.`**`drawClosedPath`**(*pointList*)

Draws a closed series of lines on the current `Layer` with the current `Brush`. No interpolation is applied to these point and *`drawLine()`* will be used to connect all the points lineraly. Coordinates are relative to the original layer size WITHOUT downsampling applied.

Parameters **`pointList`** – A list of point like `[(0, 0), (100, 100), (100, 200)]`.

Return type Nothing.

`Paynter.`**`drawRect`**(*x1*, *y1*, *x2*, *y2*, *angle=0*)

Draws a rectangle on the current `Layer` with the current `Brush`. Coordinates are relative to the original layer size WITHOUT downsampling applied.

Parameters

- **x1** – The X of the top-left corner of the rectangle.
- **y1** – The Y of the top-left corner of the rectangle.
- **x2** – The X of the bottom-right corner of the rectangle.
- **y2** – The Y of the bottom-right corner of the rectangle.
- **angle** – An angle (in degrees) of rotation around the center of the rectangle.

Return type Nothing.

`Paynter.`**`fillLayerWithColor`**(*color*)

Fills the current `Layer` with the current `Color`.

Parameters **`color`** – The `Color` to apply to the layer.

Return type Nothing.

`Paynter.`**`addBorder`**(*width*, *color=None*)

Add a border to the current `Layer`.

Parameters

- **width** – The width of the border.
- **color** – The `Color` of the border, current `Color` is the default value.

Return type Nothing.

2.1.4 `Layer` operations

`Paynter.`**`newLayer`**(*effect=''*)
Creates a new `Layer` to the current `Image`.

Parameters **`effect`** – A string with the blend mode for that layer that will be used when during the rendering process. The accepted values are: `'soft_light','lighten', 'screen','dodge','addition','darken','multiply','hard_light', 'difference','subtract','grain_extract','grain_merge','divide', 'overlay'`.

Return type Nothing.

`Paynter.`**`setActiveLayerEffect`**(*effect*)
Changes the effect of the current active `Layer`.

Parameters **`output`** – A string with the one of the blend modes listed in *`newLayer()`*.

Return type Nothing.

`Paynter.`**`duplicateActiveLayer`**()
Duplicates the current active `Layer`.

Return type Nothing.

2.2 The `Brush` Module

The second most important module of the PaYnter project. This module will manage all the data relative to the brushes you will create and use inside your scripts.

The *`Brush`* is a quite complex class in the insides, but luckly the Paynter manages all the drawing. So the final user of the library needs only to instantiate one (or more) brush and pass it to his `Paynter`.

2.2.1 The `Brush` Class

`class` `paynter.brush.`**`Brush`**(*tipImage*, *maskImage*, *size=50*, *angle=0*, *spacing=1*, *fuzzyDabAngle=0*, *fuzzyDabSize=0*, *fuzzyDabHue=0*, *fuzzyDabSat=0*, *fuzzyDabVal=0*, *fuzzyDabMix=0*, *fuzzyDabScatter=0*, *usesSourceCaching=False*)
The *`Brush`* class is the one that defines how your Paynter should draw his lines. To create this class you can use the default constructor.

```
from paynter import *
pixelBrush = Brush( "pixel.png", "", size = 100, spacing = 1)
```

`Brush.`**`__init__`**(*tipImage*, *maskImage*, *size=50*, *angle=0*, *spacing=1*, *fuzzyDabAngle=0*, *fuzzyDabSize=0*, *fuzzyDabHue=0*, *fuzzyDabSat=0*, *fuzzyDabVal=0*, *fuzzyDabMix=0*, *fuzzyDabScatter=0*, *usesSourceCaching=False*)
The creation of a *`Brush`* can be as simple as the example above or pretty complex. The complexity is based on what effect you want to obtain from your Brush. There are a lot of parameters that you can define to customize your Brush the way you want to be.

This is an example of a complex brush that emulates a watercolour brush:

```
watercolor = Brush(["watercolor1.png","watercolor2.png","watercolor3.png",
↪"watercolor4.png","watercolor5.png"], # Use more than a single brush tip image␣
↪to create more randomization
                        "",                             # No texture for this␣
↪brush
                        size = 440,                     # Quite big brush
                        angle = 0,                      # No angle since we␣
↪randomize it later each dab
                        spacing = 0.5,                  # Spacing to half the␣
↪size of the brush
                        fuzzyDabAngle = [0, 360],       # Randomize angle each dab
                        fuzzyDabSize = [1, 2],          # Randomize the size of␣
↪each dab between original size and double the size
                        fuzzyDabHue = [-0.03, 0.03],    # Randomize slightly the␣
↪hue of each dab
                        fuzzyDabSat = [-0.2, 0.2],      # Randomize slightly the␣
↪saturation of each dab
                        fuzzyDabVal = [-0.1, 0.1],      # Randomize slightly the␣
↪value of each dab
                        fuzzyDabMix = [0.4, 0.5],       # Randomize slightly the␣
↪mix of each dab
                        fuzzyDabScatter = [0, 100])     # Make each dab go a bit␣
↪of the trail to create more randomization
```

Parameters

- **tipImage** – A string with the path of the image that you want to use as a tip. If you want to load multiple brush tips and randomize between them each dab, you can place a list of strings instead.
- **maskImage** – A string with the path of the image that you want to use as a texture to apply to your brush.
- **size** – An integer with the size in pixel of the brush.
- **angle** – An integer with the angle (degrees) of the brush.
- **spacing** – A float with the spacing of the brush. Spacing is tells the system how far apart are each brush dab is. The value of this parameter is proportional to the brush size.
- **fuzzyDabAngle** – A list of two integers with the range between randomize the brush angle (degrees) of each dab.
- **fuzzyDabSize** – A list of two numbers with the range between randomize the brush size (in a proportional way to the current brush size) of each dab.
- **fuzzyDabHue** – A list of two float with the range between randomize the hue of each dab.
- **fuzzyDabSat** – A list of two float with the range between randomize the saturation of each dab.
- **fuzzyDabVal** – A list of two float with the range between randomize the value of each dab.
- **fuzzyDabMix** – A list of two float with the range between randomize the mix of each dab.
- **fuzzyDabScatter** – A list of two numbers with the range between randomize the deviation from the actual coordinates of each dab.
- **usesSourceCaching** – A boolean telling the system if he can cache the brush dab for much faster consecutives dab. Not advised when the brush has fuzzy dab parameters.

2.3 The `Image` Module

Another key class inside the Paynter library is the Image class. This module will take care of storing all the layers data of the current image you are creating. It will also manage to merge all the layer during the final rendering process.

This class will normally be 100% managed through the :py:class‘Paynter‘ class, so you should never instantiate it manually.

2.3.1 The `Image` Class

class paynter.image.**Image**

The *Image* class is structured as an array of `Layer`.

An Image class is created when you create a `Paynter`, so you can access this class as follows:

```
from paynter import *

paynter = Paynter()
image = paynter.image
```

`Image`.**getActiveLayer**()

Returns the currently active `Layer`.

Return type A `Layer` object.

`Image`.**newLayer**(*effect=''*)

Creates a new `Layer` and set that as the active.

Parameters **effect** – A string with the blend mode for that layer that will be used when during the rendering process. The accepted values are: `'soft_light'`,`'lighten'`, `'screen'`,`'dodge'`,`'addition'`,`'darken'`,`'multiply'`,`'hard_light'`, `'difference'`,`'subtract'`,`'grain_extract'`,`'grain_merge'`,`'divide'`, `'overlay'`.

Return type Nothing.

`Image`.**duplicateActiveLayer**()

Duplicates the current active `Layer`.

Return type Nothing.

`Image`.**mergeAllLayers**()

Merge all the layers together.

Return type The result `Layer` object.

2.4 The `Layer` Module

This is the class that compose the `Image` objects. This module will take care of storing data as 3D arrays of uint8. The array dimension are width, height, and RGBA channels of that layer.

This class will be 100% managed through the :py:class‘Paynter‘ class, so you should never instantiate it manually.

2.4.1 The `Layer` Class

class paynter.layer.**Layer**(*data=None*, *color=None*, *effect=''*)

The *Layer* class contains a 3D array of N.uint8 and a string with the blend mode of the layer.

An Image starts with one layer inside, but you can create more of them as follows:

```
from paynter import *

#Inside the paynter there is already an Image with a blank Layer.
paynter = Paynter()

#Create a blank new layer
payner.newLayer()

#Create a new layer duplicating the current one
payner.duplicateActiveLayer()

#Gets the current active layer
layer = paynter.image.getActiveLayer()
```

Layer.**showLayer**(*title=''*, *debugText=''*)

Shows the single layer.

Parameters

- **title** – A string with the title of the window where to render the image.
- **debugText** – A string with some text to render over the image.

Return type Nothing.

2.5 The `Color` Module

The Color class is another fundamental class in the Paynter library. This module will mange the creation, modification, and storing of colors and palettes.

The color class is mainly used internally by the `Paynter` class, but the user will still have to create the palette and sets the active colors manually through Paynter.setColor(color).

2.5.1 The `Color` Class

class paynter.color.**Color**(*r*, *g*, *b*, *a*)

The *Color* class has 4 0-1 floats, one for each RGBA channel.

A Color class is created from palette functions or directly with their constructor.

```
from paynter import *

#Get a list of colors
palette = getColors_Triad(spread = 20)

#You can create with 0-1 floats..
otherColor = Color(1, 0.5, 0, 1)

#.. or with 0-255 ints
sameColor = Color(255, 128, 0, 255)
```

Color.**get_0_255**()
Gets the RGBA 0-255 rappresentation of this color.

Return type A list of 4 ints.

Color.**get_0_1**()
Gets the RGBA 0-1 rappresentation of this color.

Return type A list of 4 floats.

Color.**get_HSV**()
Gets the HSV 0-1 rappresentation of this color. This does ignore the alpha value.

Return type A list of 3 floats.

Color.**set_HSV**(*h*, *s*, *v*)
Overwrite the current color with this set of HSV values. This keeps the current alpha value.

Parameters

- **h** – A 0-1 float with the Hue.
- **s** – A 0-1 float with the Saturation.
- **v** – A 0-1 float with the Value.

Return type The new *Color* object.

Color.**set_alpha**(*newAlpha*)
Overwrite the current alpha value.

Parameters **newAlpha** – A 0-1 float with the desired alpha.

Return type The *Color* object.

Color.**get_alpha**()
Gets the current alpha value.

Return type A 0-1 float.

Color.**tweak_Hue**(*ammount*)
Change the current hue value by a certain ammount.

Parameters **ammount** – A 0-1 float with the ammount to sum to the current hue.

Return type The *Color* object.

Color.**tweak_Sat**(*ammount*)
Change the current saturation value by a certain ammount.

Parameters **ammount** – A 0-1 float with the ammount to sum to the current saturation.

Return type The *Color* object.

Color.**tweak_Val**(*ammount*)
Change the current value value by a certain ammount.

Parameters **ammount** – A 0-1 float with the ammount to sum to the current value.

Return type The *Color* object.

Color.**copy**()
Creates a copy of this Color.

Return type The new *Color* object.

CHAPTER 3

About

3.1 Goal

The goal for the project is to have a simple yet really powerful interface for all the people who wants to procedurally generate images without letting down all the features that you have inside a real image painting software.

3.2 Contacts

If you want to contact be for any reason, please feel free to do so at elkiwydev@gmail.com

3.3 License

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

F

G

I

L

M

N

P

R

S

T