This package defines macros to draw pixel-art pictures using Lua\LaTeX{}.
Part I.
Introduction

Installation instruction are given in section 3. Documentation about how to use this package (and examples) is given in part II. Most of the examples has been gathered in appendix A.

1. Note to non-Lua\LaTeX\ users

Unfortunately, package pixelart does not works with Lua\LaTeX\ or Xe\LaTeX. If you cannot or don’t want to use Lua\LaTeX, you can:

- use the previous, unmaintained version of pixelart, named pixelart0, and included with this package;
- or, better, use pxpic.

2. License

This work may be distributed and/or modified under the conditions of the \LaTeX\ Project Public License, either version 1.3 of this license or (at your option) any later version.

3. Installation

3.1. Gnu/Linux Distribution

If applicable, the easiest way to get pixelart working is by installing it by your distribution package. In Debian (and Ubuntu, and surely other distributions that inherit from Debian) it is packaged in texlive-pictures since version 2017.20180103-1. So you can install it by running:

`sudo apt install texlive-pictures`

3.2. \LaTeX\ distribution

This package is included both in TeXLive and MiKTeX. It can be installed using their respective package managers.
3.3. Manual installation

- Download the lastest archive:
  
  **Stable version** https://mirrors.ctan.org/graphics/pgf/contrib/pixelart.zip
  
  **Development version** https://framagit.org/spalax/pixelart/repository/archive.zip?ref=main

- Unzip the archive.

- Move files `pixelart.sty` and `pixelart.lua` into a \LaTeX path.

Part II.

Usage

4. Package options

The package accepts a single option.

draft

Loading the package with this option is equivalent to using option `draft` on every single call of command `\pixelart` (or `\tikzpixelart`). If your document contains a lot of those commands, using this option can speed up compilation. For instance, on my outdated computer, compilation of this document takes about 11 seconds with this option, and more than 16 seconds without.

See option `draft` (page 4) for more information.

5. Drawing pictures

`\pixelart[{options}]{{pixels}}`

This is the basic command to draw pixel-art. Its mandatory argument is a sequence of pixels, as characters which will be turned into colored squares (using a `tikzpicture`, from Till Tantau’s \LaTeX). Its basic usage is given in figure 1 (page 4).

In argument `{{pixels}}`, dots . are converted to transparent pixels, spaces and line breaks delimit pixel rows, and any character defines a color, depending on option `colors`.

More information is given in section 5.1.

pixels

The art, as a list of pixels. See section 5.1.

`colors = {{options}}`

Pixel colors. See section 5.2.
5. Drawing pictures

\begin{figure}[ht]
\centering
\begin{verbatim}
\\pixelart{
\begin{tikz}
\red,
scale=.05,
baseline=.3em,
\}
\}
\end{tikz}
\end{verbatim}
\caption{Basic example}
\end{figure}

draft = \texttt{true}|false

Disable rendering of \pixelart and \tikzpixelart commands: on documents containing a lot of those commands, this can speed up compilation. The pixel art is replaced by a dummy drawing, which takes the same space as the pixel art it replaces.

This option can be applied to every single \pixelart or \tikzpixelart of the document by using \usepackage[draft]{pixelart}.

See an example in A.5.

\texttt{margin = \langle length \rangle}

Pixel-art pictures are \textit{enclosed} inside a \texttt{tikz clip} command. Most of the time, you won’t notice it, but when playing with options \texttt{style} or \texttt{tikz}, your picture might extend outside usual borders, and part of it might be clipped out. Use the \texttt{margin} option to extend the clip area. See example A.13.

\texttt{squares}

If set, each pixel is drawn using a separate \texttt{fill} command (more in section 5.3). This option is incompatible with \texttt{stack} (see section 5.5).

\texttt{stack}

If set, \pixelart will try to draw several adjacent pixels of the same color using a single \texttt{fill} command (more in section 5.4). This option is incompatible with \texttt{squares} (see section 5.5).

\texttt{style = \langle keyword \rangle}

Default: \pixelart
5. Drawing pictures

Pixels are drawn using \texttt{\textbackslash fill} commands in a \texttt{tikzpicture} environment. Those \texttt{\textbackslash fill} commands are applied the \texttt{pixelart} style by default (as in: \texttt{\textbackslash fill[pixelart] (0, 0) rectangle (1, 1);}). You can set another style by using this command. See an example in A.6.

\texttt{tikz = \{⟨options⟩\}}

A pixel art is a bunch of \texttt{\textbackslash fill} commands in a \texttt{tikzpicture}. Arguments to this option are passed as-is as optional arguments to the \texttt{tikzpicture} environment. See examples in A.7.

You will probably use this option to make the pixel art smaller (it is, by default, probably bigger than what you want), and to change the baseline. See figure 1 for an example.

Note that a \texttt{\textbackslash clip} command, which does not accept arguments, is used to delimit the picture. So you might get an error if your \texttt{tikz} option applies to \texttt{\textbackslash clip} commands (for instance, by using \texttt{every path/.style=bla bla bla}, which would apply to the \texttt{\textbackslash clip} command, and would raise an error). In such cases, you might want to use option \texttt{style} instead (see page 5).

\texttt{\textbackslash tikzpixelart[⟨options⟩]{⟨coord⟩}{⟨pixels⟩}}

Use this command if you want to draw pixel art pictures inside a \texttt{tikzpicture} environment.

The \{⟨options⟩\} and \{⟨pixels⟩\} arguments have the same meaning as the corresponding arguments to the \texttt{pixelart} command (excepted for \texttt{tikz}, which is passed as is as the argument of a scope environment). The \{⟨coord⟩\} is the coordinate of the bottom left corner of the pixel art. It is passed as is in a TikZ scope environment (\texttt{\begin{scope}[shift={<coord}>]}), so you can use anything that will be correctly parsed by TikZ.

See examples in section A.10.

5.1. Pixels

Pixels are defined as blocks of characters. Dots always represent transparent pixels, and spaces and line breaks always separate pixel rows. Then, other characters encode different colors, depending on option \texttt{colors}.

Let us analyse the following example.

\begin{verbatim}
\begin{pixelart[
  \texttt{\textbackslash colors=RGB, tikz={scale=.4}}
]{
  \{yellow\}GB
  G.R
  BR{}
}\end{pixelart}
\end{verbatim}

- The first row \texttt{\{yellow\}GB} means: one yellow pixel, one green pixel, one blue pixel.
5. Drawing pictures

- The second row G.R means: one green pixel, one transparent pixel, one red pixel.
- The last row BR{} means: one blue pixel, one red pixel, one pixel using the default TikZ color (that is, a \fill command is used to draw this pixel, without any color option, which means the default TikZ color is used; see option tikz, page 5).

Note that:

- Line breaks and consecutive spaces are considered as a single row separation. That is, the previous example could also have been written as \{yellow\}B G.R BR{}.
- New paragraph are not allowed inside a \{⟨pixels⟩\} argument.
- If rows do not have the same length (the same number of pixels), transparent pixels are added at the end of the small rows.

Examples for the previous corner cases are given in section A.1.

Most of commands are expanded before being passed as an argument to pixelart, so it is possible to use commands inside a pixelart{⟨pixels⟩} argument. See section A.2 for an example.

5.2. Colors

In a \pixelart command, pixel colors can be defined explicitly, or using keys/values.

**Explicit** Any text surrounded by braces is transmitted as-is to the underlying tikzpicture. That way, you can use anything you would use with TikZ (which itself uses xcolor): \textcolor{red}{red}, \textcolor{red!20}{red!20}, \textcolor{red!20!purple}{red!20!purple}, any user-defined color, or \{}, which does not give any color argument to the \fill command used to draw this pixel, thus using default TikZ color (see option tikz, page 5).

**Key/Value** Any alphanumeric character (a to z, A to Z, 0 to 9) not surrounded by braces is a key, which encodes the corresponding color defined in the colors argument. For instance, if colors is given as \texttt{colors={R={red},G={green}}} in the \{⟨pixels⟩\} argument is equivalent to \{\red\{green\}⟩.

Using non-alphanumeric characters as keys of colors is not supported: it might work, but it is an undefined behaviour.

The following examples illustrates this option.

- Key/values:
5. Drawing pictures

\pixelart[
  colors={R=red, G=green, B=blue}
]{
  RG
  GB
}

- Explicit:

\pixelart{
  {cyan}{magenta}
  {yellow}{black}
}

- Mix:

\pixelart[
  colors={R=red, G=green, B=blue}
]{
  RG
  {yellow}B
}

If you use the same setting several times, you can define it once for all using `\newpixelartcolors`.

`\newpixelartcolors{\langle name\rangle}{\langle key/value\rangle}`

Once a color set has been defined, to use it, simply use its name as an argument to `colors` instead of the whole keys/values. See an example in A.3.

**name**
Name of your color set.

**key/value**
Your color set, as keys/values.

`\renewpixelartcolors{\langle name\rangle}{\langle key/value\rangle}`

The previous command will fail if a color set `{\langle name\rangle}` is already defined. This command will always succeed, whether a color set `{\langle name\rangle}` already exists or not.
5. Drawing pictures

The following key/value settings are predefined (examples are given in A.4).

**explicit**: No keys/values: pixel colors have to be explicitly defined.

```
\newpixelartcolors{explicit}{}
```

**RGB**: Define the primary colors red, green, blue, as well as black and white.

```
\newpixelartcolors{RGB}{
  R=red, 
  G=green, 
  B=blue, 
  W=white, 
  K=black
}
```

**BW**: Black and white.

```
\newpixelartcolors{BW}{
  0=white, 
  1=black, 
}
```

**gray**: Gray scale: 0 is white, 9 is black, and digits 0 to 9 are a gray scale from white to black.

```
\newpixelartcolors{gray}{
  0=white, 
  1=white!89!black, 
  2=white!78!black, 
  3=white!67!black, 
  4=white!56!black, 
  5=white!44!black, 
  6=white!33!black,
}
```
5. Drawing pictures

![Image](image.png)

**Figure 2:** Artifacts appearing with option `squares`: the white lines should not exist.

```latex
\begin{verbatim}
\newpixelartcolors{mono}{
    a={},
    b={},
    c={},
    9={},
}
\end{verbatim}
```

**mono**: Monochromatic: any character encodes the default TikZ color (black by default, may be changed with options `tikz` or `style`).

### 5.3. More about option `squares`

When using `squares`, each pixel is a separate colored square. Thus, artifacts can appear when viewing the document, as illustrated in figure 2.

To remove those artifacts, you can (1) use option `stack` instead (see section 5.4), or (2) use option `overlap`.

**overlap = \{\textit{number}\}**

Assuming the square width is 1, `\{\textit{number}\}` is the size of the overlap: how much a square overlaps over the next one. With a positive number, the artifact (white line between two pixels) will disappear as the adjacent pixels are merged.

Note that using a negative value to `overlap` is also allowed, and will separate each pixel from its neighbour, giving a the look of a LCD screen.

See examples in section A.9.
6. Default options

<table>
<thead>
<tr>
<th>Drawing</th>
<th>mono</th>
<th>concentric</th>
<th>checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixelart0</td>
<td>243</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pixelart stack</td>
<td>2.9</td>
<td>4.4</td>
<td>158</td>
</tr>
<tr>
<td>pixelart squares</td>
<td>245</td>
<td>263</td>
<td>252</td>
</tr>
</tbody>
</table>

(a) File size (kB)

<table>
<thead>
<tr>
<th>Drawing</th>
<th>mono</th>
<th>concentric</th>
<th>checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixelart0</td>
<td>38.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pixelart stack</td>
<td>1.0</td>
<td>1.0</td>
<td>24.4</td>
</tr>
<tr>
<td>pixelart squares</td>
<td>19.9</td>
<td>28.3</td>
<td>27.1</td>
</tr>
</tbody>
</table>

(b) Compilation time (seconds)

Figure 3: Benchmark: pixelart0 vs. stack vs squares

5.4. More about option stack

If option stack is set, pixelart tries to merge adjacent pixels of the same color, to draw them at once. Thus, it mostly remove the artifacts problem discussed with option squares (although it may still occur).

Example in section A.8 shows how different color shapes are stacked onto each other to produce the pixel art.

5.5. Which option: squares or stack?

Which option should you use: squares or stack?

To compare the different algorithms, three files have been compiled, each with one big pixel art command (see example A.11), with three different algorithms: outdated package pixelart0, and options stack and squares of pixelart. The result can be seen in figure 3 (page 10).

Unless you want to use option overlap of option squares (see section 5.3), you probably want to use option stack:

- it is faster (see benchmark);
- it produces smaller files (see benchmark);
- it mostly remove artifacts.

Thus, the stack option is the default one used by pixelart.

6. Default options

You can use the following command to define options once for all.
7. Logos

\setpixelartdefault{⟨options⟩}
Any options defined here will affect later \pixelart or \tikzpixelart commands, unless those options are explicitely set.
See example in section A.12.

7. Logos

If you want to credit \pixelart in a fancy way, you can use one of those logos. Apart from being gorgeous, they include a transparent text (so that they are searchable and copyable) and they scale with the current font size.

\pixelartlogo \ PixelArt Name and heart, in color.
\pixelartlogobw \ PixelArt Name and heart, black and white.
\pixelartheart \ Heart only, in color.
\pixelartheartbw \ Heart only, black and white.
\pixelartname \ PixelArt Name only, in color.
\pixelartnamebw \ PixelArt Name only, black and white.

Note that those logos are affected by \setpixelartdefault, so, if you have used this command in your document, you might want to call \setpixelartdefault{} to reset the default options before drawing any logo.

8. Debugging

What if drawing your pixel art throws an error?

1. Standard output prints one % pixelart 2, file ./foo.tex, input line 19 per \pixelart or \tikzpixelart command. If compilation failed right after this line, look at the given file: the \pixelart command might be wrong.

2. If you do not see the problem, set debug mode on.

\setpixelartdebugon \ Set debug on.
\setpixelartdebugoff \ Set debug off.

Using those commands, the following piece of information will be logged:

- the parsed options of \pixelart and \tikzpixelart;
- the ⟨⟨pixels⟩⟩ argument of those commands;
- the tikz commands used to draw the picture;
- and maybe some other stuff I forgot.
Note that you can have as many `\setpixelartdebugon` and `\setpixelartdebugoff` commands as you want, even if debug is already on or off.

A good idea is to copy the tikz commands that are logged (those are the commands used to draw the pixel art), and paste them in a new file. Try to compile it. Are some of your arguments at fault? Bad value for the tikz option can produce faulty tikz code.

3. Still no idea why your `\pixelart` command fails? Seek help on your usual places, or report a bug.

Part III.
Appendix

A. Examples

A.1. Corner cases of `\pixelart{pixels}` argument

- Line breaks and multiple spaces are considered as a single row separation.

  ```latex
  \begin{verbatim}
  \pixelart{
    1.1
    .1.
    1.1
  }
  \pixelart{
    1.1
    .1.
    1.1
  }
  \end{verbatim}
  
  - Multiple line breaks (new paragraphs) are not supported: at best, they do not give the expected result; at worst, compilation will fail.
A. Examples

- Empty pixels are automatically added at the end of shorter lines.

\begin{verbatim}
\pixelart{
  .1
  1.
}
\end{verbatim}

A.2. Commands inside $\texttt{\textbackslash pixelart}(\texttt{pixels})$ argument

If we are too lazy to repeat every single pixel to draw a checker board, we can use commands.

Example 1: Each couple of lines is repeated four times.

\begin{verbatim}
\newcommand{\twolines}{
  .1.1.1.1 1.1.1.1.1%
}
\pixelart{
  \twolines
  \twolines
  \twolines
  \twolines
}
\end{verbatim}
A. Examples

Example 2: \texttt{\texttt{\textbackslash foreach} outside \texttt{\textbackslash pixelart}}

```
\texttt{\begin{verbatim}
\foreach \ j in \{1, ..., 3\} {
  \xdef\line{\line.1}
}\foreach \ i in \{1, ..., 4\} {
  \xdef\board{\board \line. \line.}
}\pixelart{\board}
\end{verbatim}}
```

Example 3: I do not understand \LaTeX{} enough to explain why, however, \texttt{\texttt{\textbackslash foreach} inside a \texttt{\textbackslash pixelart} does not work.}

```
\texttt{\begin{verbatim}
\pixelart{
  \foreach \ i in \{1, ..., 8\} {
    \foreach \ j in \{1, ..., 3\} \line.1 \line.1}
}\end{verbatim}}
```

A.3. Define new color sets: \texttt{\texttt{\textbackslash newpixelartcolors}}

- Colors may be defined in the \texttt{colors} option

```
\texttt{\begin{verbatim}
\definecolor{purple1}{RGB}{247, 176, 207}
\definecolor{purple2}{RGB}{238, 33, 120}
\definecolor{purple3}{RGB}{214, 45, 117}
\definecolor{purple4}{RGB}{217, 25, 92}
\definecolor{purple5}{RGB}{173, 29, 69}
\pixelart[colors={
  1=purple1, 2=purple2, 3=purple3, 4=purple4, 5=purple5,}
}]
\end{verbatim}}
```
A. Examples

Colors may be defined using \texttt{\newpixelartcolors}

\begin{verbatim}
definecolor{purple1}{RGB}{247, 176, 207}
definecolor{purple2}{RGB}{238, 33, 120}
definecolor{purple3}{RGB}{214, 45, 117}
definecolor{purple4}{RGB}{217, 25, 92}
definecolor{purple5}{RGB}{173, 29, 69}
\newpixelartcolors{purple}{
  1=purple1, 2=purple2, 3=purple3, 4=purple4, 5=purple5,
}
\pixelart[colors=purple]{
  .54.43.
  4224215
  3445435
  .52424.
  .434..
  ...2...
}
\end{verbatim}

A.4. Predefined color sets

\texttt{mono}
A. Examples

\begin{verbatim}
\pixelart[colors=mono, tikz={
  red, scale=.3}]{
  ..01..23..
  .456789ab.
  cdefghijkl
  mnopqrstuvwxyz
  wxyzABCDEF
  .GHIJKLMNOP
  ..OPQRSTUVWXYZ
  ...UVWXYZ
  ....YZ....
}\end{verbatim}

\textbf{gray}

\begin{verbatim}
\pixelart[colors=gray, tikz={
  scale=.3}]{
  987654
  876543
  765432
  654321
  543210
  43210.
}\end{verbatim}

\textbf{BW}

\begin{verbatim}
\pixelart[colors=BW, tikz={
  scale=.3}]{
  01010101
  10101010
  01010101
  10101010
  01010101
  10101010
  01010101
  10101010
  01010101
  10101010
}\end{verbatim}
A. Examples

RGB

\[
pixelart[colors=RGB, tikz={scale=.1}]{KKKKKKK
.RRR...GG..BBB..K..K.KWKKWK
.RRR..G...BBB..KK...KWKKWK
.R..R..GG..BBB..K..K.KWKWK}
\]

A.5. Option draft

\[
\setpixelartdefault{tikz={scale=.3}}
pixelart[colors=RGB]{RRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR}
pixelart[colors=RGB, draft]{RRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR
.RRRRRRRRR}
\]
A. Examples

A.6. Option style

1 \pixelart{
2 .1
3 1.
4 }

1 \pixelart[tikz={pixelart/.style={
red}}]{
2 .1
3 1.
4 }

1 \tikzset{pixelart/.style={blue,
draw=orange, very thick, rounded
corners=5}}
2 \pixelart{
3 .1
4 1.
5 }

1 \pixelart[tikz={pixelart/.style={
green}}]{
2 .1
3 1.
4 }

1 \tikzset{myfancystyle/.style={
yellow}}
2 \pixelart[style=myfancystyle]{
3 .1
4 1.
5 }

18
A. Examples

A more complex example.

```
\tikzset{pixelart/.style={cyan, very thick, draw=black, rounded corners=5}}
\pixelart[squares={overlap=-.2}, colors=mono, tikz={scale=.5}, margin=.1]{
  1..1.1..1
  1..1.....1
  1111...1
  1..11...
  1..1..1..1
}
```

A.7. Option `tikz`

```
\pixelart[stack, colors={Y=yellow, K=black}, tikz={scale=.5, rounded corners=5}]{
  ..YYYY..
  .YYYYYYY.
  YYKYYYYY
  YYYYYYYY
  YYYYYYYK
  YYKYYYYY
  .YYYYYY.
  ..YYYY..
}
```

A.8. Drawing steps with the `stack` option

Here is how an image is built with the `stack` option: each of the following steps is a `\fill` command with carefully crafted shape. Those filled area are “stacked” on each other, to produce the expected result.

Line per line, the algorithm checks if the pixels have been drawn yet. In the following steps, the gray cross marks the first pixel that was not drawn yet, that triggered a new `\fill` command.
A. Examples

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Step 2</th>
<th>Step 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 4</th>
<th>Step 5</th>
<th>Step 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

A.9. Option overlap

\begin{verbatim}
\setpixelartdefault{tikz={scale=.3}}
\newcommand{\three}{
  111.
  11.11
  ...11
  ...11
  ...11
  ...11
  ...11
  ...11
  11.11
  111.}
\pixelart[squares={overlap=.1}]{\three}
\pixelart[squares={overlap=0}]{\three}
\pixelart[squares={overlap=-.1}]{\three}
\end{verbatim}
A. Examples

A.10. Command \tikzpixelart

\begin{tikzpicture}
\fill[red] (0, 0) circle (1);
\tikzpixelart[tikz={color=green}]{(-1, -1)}{
\hspace{1}\hspace{1}
\hspace{1}\hspace{1}
\hspace{1}\hspace{1}
\hspace{1}\hspace{1}
\hspace{1}\hspace{1}
\hspace{1}\hspace{1}
\hspace{1}\hspace{1}}
\end{tikzpicture}

A.11. Pictures used for the benchmark

The benchmark discussed in section 5.5 (page 10) uses three files with three big \pixelart commands. A small version of those commands is shown below.

\begin{verbatim}
\pixelart{
KKKKKKKK
KKKKKKKK
KKKKKKKK
KKKKKKKK
KKKKKKKK
KKKKKKKK
KKKKKKKK
KKKKKKKK
}
\end{verbatim}

\textbf{mono} Every single pixel of the same color.

\textbf{concentric} Centenric squares, that is, several area with adjacent pixels of the same color.
A. Examples

\begin{figure}
\begin{center}
\begin{Verbatim}
A. Examples

A.1. checker
\begin{verbatim}
A (kind of) checker board, where no two adjacent pixels have the same color.

\begin{verbatim}
\setpixelartdefault{colors=mono,
tikz={scale=.2, red}}
\end{verbatim}
\end{verbatim}

A.12. Default options

\begin{verbatim}
\setpixelartdefault{colors=mono,
tikz={scale=.2, red}}
\end{verbatim}

From now on, pixel art pictures will be small, and red: \begin{verbatim}
\pixelart{1. .1}.
\end{verbatim}

Excepted this one: \begin{verbatim}
\pixelart[
  tikz={scale=.4, blue}
]{}{1. .1}.
\end{verbatim}

Back to normal: \begin{verbatim}
\pixelart{1. .1}.
\end{verbatim}

\end{center}
\end{figure}
B. Change History

A.13. Option margin

In the first picture, you can see that part of the orange line has been clipped away. In the second one, with option margin correctly set, it is not clipped away.

```latex
1 \setpixelartdefault{squares}
2 \tikzset{pixelart/.style={
3 draw=orange,
4 rounded corners=5,
5 line width=5,
6 }}
7 \pixelart{11 11}
8 \pixelart[margin=.1]{11 11}
```

B. Change History

This is a raw copy of the CHANGELOG.md file that can be found in the git repository of pixelart.

- pixelart 1.0.2 (2023-02-20)
  - Fix error while trying to load pixelart.lua (closes #2).
  - Louis Paternault spalax@gresille.org¹

- pixelart 1.0.1 (2023-02-18)
  - Bugs
    * Minor change to take into account backward-incompatible change in luakeys v0.13.0 (thanks Jonathan P. Spratte).
    * Pixelart (with option square) no longer fails when asked to draw an empty pixelart.
  - Documentation
    * Minor changes and improvements.
    - Louis Paternault spalax@gresille.org²

- pixelart 1.0.0 (2022-11-16)
  - Full, backward incompatible, rewrite of pixelart, in Lua.
  - Keep version 0 of pixelart as pixelart0.

 ¹. mailto:spalax@gresille.org

². mailto:spalax@gresille.org
B. Change History

- Louis Paternault spalax@gresille.org³

• pixelart 0.3.0 (2022-11-16)
  - Renamed pixelart to pixelart0.
  - Mark pixelart0 as unmaintained.
- Louis Paternault spalax@gresille.org⁴

• pixelart 0.2.0 (2018-02-25)
  - Add a draft package option.
- Louis Paternault spalax@gresille.org⁵

• pixelart 0.1.2 (2018-01-13)
  - Fix bug: First line-break is now automatically ignored.
  - Minor documentation fixes and improvements.
- Louis Paternault spalax@gresille.org⁶

• pixelart 0.1.1 (2017-12-08)
  - Fix CTAN URLs.
- Louis Paternault spalax@gresille.org⁷

• pixelart 0.1.0 (2017-12-05)
  - First published version.
- Louis Paternault spalax@gresille.org⁸
C. Index

C
colors ........................................... 3, 5 ff., 14

D
draft ............................................. 3 f.

L
lppl ............................................. 2

M
margin ........................................... 4, 23

N
\newpixelartcolors .......................... 7 ff., 15

O
overlap ......................................... 9 ff.

P
\pixelart ...................................... 3–7, 12–23
pixelart (package) ............................ 2
pixelart0 (package) ......................... 2
\pixelartheart ................................ 11
\pixelartheartbw .............................. 11
\pixelartlogo ................................ 11
\pixelartlogobw .............................. 11
\pixelartname ................................ 11
\pixelartnamebw .............................. 11
pixels .......................................... 3
\pixelartnamebw (package) .................. 2
\renewpixelartcolors ....................... 7
\setpixelartdebugoff ....................... 11
\setpixelartdebugon ....................... 11
\setpixelartdefault ....................... 11, 17, 20, 22 ff.
squares ....................................... 4 ff.
stack .......................................... 4, 9 ff., 19
style .......................................... 4 ff., 9

T
TikZ (package) ................................ 3
tikz ............................................. 4 ff., 9, 12
\tikzpixelart ................................. 3 ff., 21

X
xcolor (package) ............................. 6