Pixelorama Documentation

Table of contents:

- Introduction
  - Before we start:
- Installation
  - Itch.io
  - GitHub Releases
  - Windows
    - Scoop
  - Linux
    - Flathub
    - Snap
  - macOS
  - Early Access - "Nightly" Versions
- User Interface Basics
  - Changing theme
- The Canvas
  - Navigation
    - Zoom
    - Pan
    - Mirror
  - Grid
    - Editing the Grid
    - Pixel Grid
  - Tile Mode
  - Second Canvas
  - Rulers and Guides
    - Editing Guides
    - Toggling Visibility
  - Transparent Background
    - Editing the transparent background
- The Timeline
  - Animation
    - FPS
    - Loop modes
    - Onion skinning
  - Layers
  - Frames
    - Frame tags
  - Cels
- Save and Export

- Saving
- Exporting
  - Export a sequence of images
  - Export a spritesheet
- Palettes
  - Palettes panel
    - Creating a new palette
      - Preset
      - Name
      - Comment
      - Width and Height
      - Create colors with alpha component
      - Get colors from
    - Editing palette metadata
      - Name
      - Path
      - Delete button
  - Palette grid
    - Tools
    - Colors movement
  - Palettes import
    - Supported formats
- Drawing
  - Drawing Tools
  - Pencil and Eraser
    - Pencil Tool Options
    - Pixel Perfect
  - Bucket
  - Shape tools
  - Color Picker
  - Mirroring
- Selecting
  - Selection Tools
  - Modifying Selections
    - Selecting All
    - Clearing Selection
    - Inverting Selection
  - Duplicating
  - Copy, Paste, Cut and Delete
  - Selection Preferences
- Shading
  - Simple Shading

    - Hue Shifting
- Tools
    - Left and right mouse buttons
    - The tools
        - The selection tools
        - Canvas navigation tools
        - Drawing tools and drawing-assisting tools
    - Tool options
        - Changing colors
- Transforming
    - Move
    - Scale
    - Rotate
    - Having An Active Transformation
        - Applying A Transformation
        - Canceling A Transformation
- Brush
    - Brush Types
    - Making Your Own Custom Brushes
        - File Brushes
        - Random Brushes
        - Project Brushes
- Cel
    - Linked cels
- Project
    - Saving a project
    - .pxo files
        - File structure
        - ZSTD compression
- Extension Basics
    - Making an Extension
        - Prerequisites
        - Process
        - extension.json
        - Main.tscn
        - Main.gd
    - Limitations and Workarounds
    - Exporting the Extension
    - Installing the Extension
    - Debugging an Extension
- ExtensionsApi
        - Description

    - Properties
    - Method Descriptions
  - GeneralAPI
    - Description
    - Method Descriptions
  - MenuAPI
    - Enumerations
    - Method Descriptions
  - DialogAPI
    - Method Descriptions
  - PanelAPI
    - Method Descriptions
  - ThemeAPI
    - Method Descriptions
  - ToolAPI
    - Method Descriptions
  - SelectionAPI
    - Method Descriptions
  - ProjectAPI
    - Method Descriptions
  - ExportAPI
    - Enumerations
  - Method Descriptions
  - SignalsAPI
    - Description
    - Method Descriptions
- Examples
    - Bare Minimum
    - Add new Tab
    - Add Menu Item
    - Add a Theme
    - Change Font
    - Project Manipulation
    - Custom Exporter
- Ways to contribute
  - Say something nice!
  - Spread the word!
  - Donations
  - Contributing Code
  - Contributing Feedback
  - Testing and reporting issues
  - Contributing Documentation

- Contributing Translations
- FAQ
  - Q: How much does Pixelorama cost?
  - Q: What platforms is Pixelorama available on?
  - Q: Who is working on Pixelorama?
  - Q: Is Pixelorama safe?
  - Q: What engine is being used to develop Pixelorama? Where is the code hosted?
  - Q: Can I contribute to this project?
  - Q: Will Pixelorama ever become Godot's Plugin?
  - Q: What languages is Pixelorama available in?
  - Q: What are the minimum system requirements to run Pixelorama?
- Compiling from Source
- Troubleshooting
  - Pixelorama crashed while I was working on my project!
  - My images are being saved as pxo files, and not png, jpg, gif, etc
  - macOS says that Pixelorama is damaged!
  - I have performance issues, Pixelorama is very slow on my machine
  - I have a hiDPI monitor and the User Interface is very small!
  - Error code 12 when exporting png files
  - Drawing is very slow when using a drawing tablet
  - Guides are not being created

Introduction

> **OFFLINE VERSION**
>
> You can also download the offline version of this documentation.

Welcome, fellow dreamer, to the documentation of Pixelorama: your free and open-source pixel-art creation tool, developed by Orama Interactive and our contributors. If you are new to this documentation, we recommend you start by learning how to install the application, then reading the User Interface Basics page to familiarize yourself with Pixelorama's workspace. The ultimate goal of this program is to be able to keep up with the user's imagination and assist them in creating any kind of pixel art they wish. This could be a cool landscape, a dope animation, game graphics, random sketches or a well-thought pixelated meme. Thank you for researching our tool!

Here, you will understand the application's aspects, features, and tools from our tutorials. Eventually, you will be able to master it and apply it to your own pixelated goals. If you are an experienced pixel artist, familiar with most of Pixelorama's functionalities, you can search for some specific features and understand them better or get some ideas on how to creatively apply techniques you may not have thought of. In any case, fasten your seat belts because we are about to start!

Before we start:

If you encounter any issue regarding the documentation, the program isn't working properly, or something isn't as clarified and you have a question regarding it, you can seek help in our community channels.

- Discord server: https://discord.gg/GTMtr8s
- GitHub Repository of Pixelorama: https://github.com/Orama-Interactive/Pixelorama/
- GitHub Repository of the Documentation: https://github.com/Orama-Interactive/Pixelorama-Docs/

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Installation

Pixelorama is available for free for all major desktop platforms, namely Windows, Linux and macOS. There is also a Web version available, which can be used directly from your browser, without downloading it. Stay tuned, because more platforms will be supported soon.

Itch.io

Pixelorama is available for free on itch.io for Windows, Linux and macOS. You can get it by clicking on "Download Now". A dialog will appear that lets you set a price for the tool. While donations are appreciated and they help with the development of the software, Pixelorama is a free software and paying is optional. After you decide whether you want to donate or not, you will be taken to the download page, where you can download a version of Pixelorama depending on your operating system.

On the itch.io page, you are also able to run Pixelorama's Web version directly through your browser.

GitHub Releases

Pixelorama is available for free on GitHub for Windows, Linux and macOS. On that page, you can also find previous versions of Pixelorama, but it is recommended to always download the most recent one.

Windows

Windows users can download Pixelorama from itch.io, GitHub Releases and Scoop. It should run on all new Windows versions from Windows 7 and on. There are three available versions, 64-bit portable, 32-bit portable and a 64-bit installer. See this page to find out whether your Operating System is 64-bit or 32-bit.

Scoop

You can get Pixelorama with Scoop, by pasting these lines in the command prompt.

```
scoop bucket add extras
```

```
scoop install pixelorama
```

Linux

Linux users can download Pixelorama from itch.io, GitHub Releases, Flathub and the Snap Store. There are two available versions, 64-bit and 32-bit. See this page to find out whether your Operating System is 64-bit or 32-bit.

Flathub

You can find Pixelorama for free on Flathub. If you have Flatpak installed, follow the instructions on that page to learn how to install and run the software.

Snap

Pixelorama is available as a Snap. If you have Snap installed, run `sudo snap install pixelorama` on your Terminal to install the software.

macOS

macOS users can download Pixelorama from itch.io and GitHub Releases, as shown above. If you encounter any issues with macOS saying that the application is damaged, see this troubleshooting page.

Early Access - "Nightly" Versions

If you want to try the latest unstable builds of Pixelorama, you can find them here, available for Windows 64-bit, Linux 64-bit, as well as macOS. There is also an early-access Web version available here.

Keep in mind that these builds are very unstable and are not recommended for production.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

User Interface Basics

Pixelorama aims for a simplistic and understandable, yet professional User Interface (from now on referred as "UI"). Optimized for quick access to everything, Pixelorama's UI looks like this:

A screenshot of Pixelorama's UI which has been edited to have its interface numbered. Note that the image above is the default look of Pixelorama (Dark Theme) after you have closed the splash screen.

Now, let's categorize the common elements of which the UI is consisted of and give everything a short description.

- **1** - The traditional top bar with menus found in most windowed applications.
- **2** - The toolbar. The area where you can select a tool to work with.
- **3** - Project tabs. This is where you will find all of your currently opened projects and switch between them by clicking on their respective tab.
- **4** - The timeline. This is where you will find the sprite's layers and frames for animation.
- **5** - A small canvas preview. There you can also preview your animation independently from your main canvas.
- **6** - The tool options. This is where you can configure some settings of your currently active tools, as well as select your left and right colors.

- **7** - The palettes. Useful for easily picking pre-set colors.

Located in the middle of your view, the canvas is the "sheet" on which you are creating your artwork. This is the main area of the workspace because this is where you will be doing all of the drawing.

Each UI element has its own detailed page in the documentation, where it is analyzed in depth.

Changing theme

By default, the dark theme is enabled on Pixelorama. You can select another by going to the Edit menu on the top bar, selecting Preferences and clicking on the `Interface` tab on the left.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

The Canvas

The canvas is the area where your drawing is located and it's also where you apply all of the tools at your disposal. While you can scroll around the canvas without limits, you cannot draw outside the canvas's boundaries, which are set by the size of the image. Of course, you can always resize the project to expand or shrink your canvas's limits.

Navigation

You can navigate around the canvas in multiple ways.

Zoom

To zoom, you can use the mouse wheel. Scroll the wheel up to zoom in, and down to zoom out. You can also use `+` on your keyboard to zoom in and `-` to zoom out. You can see the current zoom level of the canvas in the top bar of the user interface.

There is also a dedicated zoom tool, which you can use like any other tool, by pressing either the left or the right mouse button, depending on where you mapped it. The zoom tool can prove useful for users with no mouse, such as graphics tablet or touch screen users. From the tool's options, you can switch it to either zoom in or out, and you can also find two buttons.

The first one is "Fit to frame" which basically centers your canvas and makes it as big as possible without any portion of it not being visible.

The second one is 100% zoom. This basically makes the canvas its exact pixel dimensions based on your screen. For example, if your canvas is 64x64 pixels, pressing this button will make the canvas appear on a 64x64 area of your screen. This gives you an idea of how your art looks at its original dimensions.

Pan

You can pan by either holding the middle mouse button or `Space` on your keyboard and simply moving the mouse. You can also use the arrow keys to move around. Like with zoom, there is also a dedicated pan tool.

Mirror

You can mirror your view horizontally and non-destructively by going to the View menu and clicking "Mirror View", or by pressing `Shift + M`.

Grid

The grid is an important tool that aids the user in accurately drawing lines. Currently, Pixelorama has two types of grids, rectangular and isometric. The rectangular grid is what you will probably use most of the time, while the isometric grid is great if you are making isometric art, like the one pictured below.

Editing the Grid

The grid's properties can be edited in the Preferences, which is found under the Edit menu, and then selecting "Canvas" on the left side. The option includes the grid type itself (rectangular, isometric or both), the width and height of the rectangular grid, the width and height of the isometric bounds of the cells, the x and y offset of the grid, the grid color and whether to draw over Tile Mode.

To toggle its visibility, you can click on "Show Grid" in the View menu, or by pressing `Control + G`.

Pixel Grid

The Pixel Grid is a rectangular 1x1 grid that appears after a certain zoom level. You can change its color and after which zoom level it appears at the same place as the rest of the grid options.

To toggle its visibility, you can click on "Show Pixel Grid" in the View menu, or by pressing `Control + H`.

Tile Mode

If you are working with tiles, it might be useful to preview them in such a way so you can see if they repeat seamlessly. For this reason, you can enable Tile Mode, which repeats the canvas in multiple dimensions around the central area. To enable Tile Mode, you can go to the View menu, hover on Tile Mode and select what you want. "None" will disable Tile Mode, "Tiled in Both Axis" will repeat the canvas horizontally, vertically and diagonally, and the other two options will just repeat it either horizontally or vertically. It is also possible to draw in the tiled area.

Second Canvas

On the right side of the canvas, there is a grabber. Moving it to the left will reveal a second canvas. The purpose of this is to easily preview your artwork in different positions or zoom levels. As of right now, you **cannot** draw on it.

Rulers and Guides

On the top and left sides of the canvas, you can find two rulers. These serve two purposes, the first is to let the user easily and effortlessly measure distances in pixels, and the second is to create guides by clicking on them with the left mouse button and, while it's being held, dragging the cursor to the position you want your guide to be at. Guides are either horizontal or vertical lines that can be used as references to help you properly align stuff on your canvas.

Editing Guides

To move an already created guide, you can drag the guide by clicking and holding the left mouse button while your cursor is pointing at the guide but it's also **outside the canvas boundaries**. If it's not, it will simply apply your currently selected tool. If the guide is completely moved outside the canvas area, it will be removed.

You can also change their colors by going to Edit, Preferences, Canvas.

Toggling Visibility

You can also easily hide the rulers and the guides from the View menu. `Control + R` toggles the visibility of the rulers, and `Control + F` toggles the visibility of the guides.

Transparent Background

If you are working with a drawing that has transparent pixels, a checkerboard background will appear there. Its purpose is to simply show that these areas are transparent. The background itself is **not part of your drawing**. Meaning, it will not be exported along with the final image.

Editing the transparent background

You can edit the checkerboard background by going to the Edit menu, Preferences, and selecting Canvas. The options include the size of the checkers in pixels, the two colors that make up the checkerboard, and whether they follow the canvas movement and zoom level. If both "Follow Canvas Movement" and "Follow Canvas Zoom Level" are toggled on, the checkerboard matches exactly the pixels of the canvas that you draw and will pretty much act as a second grid.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

The Timeline

The timeline's main purpose is to help you with creating animations. It is composed of frames, which are used for creating animations, and layers, which help you separate your frames in different parts. Each frame is essentially a collection of multiple layers, and each individual intersection of a frame and a layer is called a cel. All frames have the same amount of layers.

Animation

To preview your animation, you can click on the `Play` button (or press `F5`) located on the top right side of the timeline. The button on the left (mapped to `F4`) is used to preview the animation backward.

FPS

On the top and far right side of the timeline, you will see a number, which has the default value of 6. This is the FPS, which stands for `Frames Per Second`. To put it simply, this is the speed of the animation and it determines how many frames are being shown for every second. Having a value of 6 FPS, for example, will display 6 frames every second the animation is running.

For more details, you can read this Wikipedia article.

Loop modes

The button located directly on the left of the FPS is used to change the animation loop mode. There are three loop modes:

- Cycle loop. When the animation reaches the final frame, it will start playing again from the first frame. This is the default option.
- Ping-pong. When the animation reaches the final frame, it begins playing backward, starting from that final frame. When it reaches the first frame, it will again start playing forward until it reaches the final frame, and so on.
- No loop. When the animation reaches the final frame, it simply stops playing.

Onion skinning

Directly on the left of the loop mode button, you will find two buttons used for onion skinning. Onion skinning is useful for animations because it's a simple and easy way to preview the previous and next frames at the same time as the frame you are currently editing. For more details, you can read this Wikipedia article.

The button on the right simply toggles onion skinning on and off, and the one on the left brings up options, like the amount of past (previous) and future (next) frames that appear, and blue-red mode, which tints the previous frames blue, and the next frames red.

> **IGNORE ONION SKINNING**
>
> If you want a layer to ignore onion skinning simply add the "_io" suffix in its name.

Layers

On the left side of the timeline, you will see the layer buttons. Clicking on a layer button will select the current layer. Double-clicking will let you change the name of that layer.

The layer buttons have three more buttons on them. The first toggles the layer's visibility. While invisible, the layer cannot be edited by the user. It will also remain invisible on export. The second locks and unlocks the layer. If the layer is locked, it cannot be edited by the user, but it does remain visible. The third button is used to automatically link the new cels that are being created. If it's toggled on, it is essentially a fast way to create linked cels.

There are 6 more buttons located on top of the layers. These, in order of appearance: add a new layer, remove the currently active layer, move the current layer up and down, clone the current layer and merge the current layer with the one directly below it.

Frames

Directly above the cels, you will notice some buttons with each one having a number. These are the frame buttons, and the number they have is their order in the animation.

Clicking on a frame button will select that frame, and right-clicking a frame button will open up a menu. The menu options include a remove frame option, clone frame, move frame to the left, move frame to the right and frame properties. Clicking on frame properties brings up a window that lets you change the duration of that frame.

> **FRAME DURATION**
>
> By default, the duration of each frame is set to 1x, which means it will take exactly as much time as the timeline FPS (frames per second). Setting it to 2x, for example, that frame will take twice as much time, while setting it to 0.5x will take half the time, meaning it will play out faster.

You can also drag and drop frame buttons to re-arrange them. This will also affect their corresponding cels.

On the top side of the timeline, you will find frame-related buttons which offer the same functionality as the frame button right-click menu, like adding, removing, cloning and moving frames. You will also find a button that is used to modify `frame tags`.

Frame tags

Frame tags are used to organize different parts of your animated sprite. Those parts can also be exported independently using frame tags.

Clicking on the button mentioned above will bring up this window.

Here you can see the list of all of the frame tags of the current project. Using the plus (+) sign, you can add more tags. A tag has a starting frame (from), an ending frame (to), a name and a color.

To edit an already existing tag, you can click on the "Edit" button next to the tag you want.

If "Animation plays only on frames of the same tag" is toggled on, then, when you preview your animation, it will only preview the frames that belong on the same tag as the currently selected frame. This is a great way to preview parts of your animation separately.

Cels

Cels are the intersection of layers and frames. For a more detailed explanation, you can read this page. To select a cel in order to draw on it, you can left-click on their respective button. The cel buttons take the most space in the timeline, and they are located to the right of the layer buttons and below the frame buttons. They also have a small preview of their image data on them.

Right-clicking on a cel button will bring up a menu, with options such as Link (or Unlink) Cel, and Delete, which clears the content of the cel, making it fully transparent.

SELECTING MULTIPLE CELS

You can select/deselect multiple cels by holding the `Control` key and left-clicking individual cels, or hold the `Shift` key and left-click a cel to also select all of the cels between the one that was clicked and the one that has been selected last.

This also works on frame and layer buttons. For example, selecting two frame buttons using `Control` or `Shift` will select all of their cels as well. The same is true for layer buttons.

DRAG AND DROP

You can drag and drop layer, frame and cel buttons to re-arrange them. Re-arranging frame and layer buttons also affects their cels.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Save and Export

You can either save or export your projects. Saving a project will create a .pxo file, which is Pixelorama's custom file format, while exporting means that one (or multiple, in case of animations) `.png`(s), an `.apng`, or a `.gif` file will be created.

> **SAVING VS EXPORTING**
>
> Basically, you should **save** your project if you intend to use it again on Pixelorama and keep all of your data, such as layers, project brushes, animation tags, etc. While, if you want to share your artwork, import it in another software such as a game engine, or view it on an image editor, you should **export** it. Of course, nothing is stopping you from doing both, which is actually the recommended approach.

Saving

To save a project, you can go to the File menu and select `Save..`, or press `Control + S`, and a window will appear. If you are using a Desktop version, you will be able to choose where you want the file to be saved. If you use the Web version, you will be asked for a file name and the `.pxo` file will be downloaded by your browser.

By selecting `Save...` again on a project you have already saved, the file name and path you chose before will still be automatically used, which means that the old file will be replaced with a new one with the changes you have made. If you'd like to save a different file, you have to use the `Save as...` option, or press `Shift + Control + S`.

Exporting

To export your project, you can go to the File menu and select `Export...`, or press `Control + E`, and a window will appear. If you are using a Desktop version, you will be able to choose where you want the file to be saved. If you use the Web version, you will be asked for a file name and the `.png`, `.apng` or `.gif` file(s) will be downloaded by your browser.

Just like with save, you have to use the `Export as...` option or `Shift + Control + E` to export to a new directory.

You can choose what you want to export by clicking on one of the tabs on the top of the window.

> **RESIZE WHILE EXPORTING**
>
> If you draw on a small canvas, the exported image will have a relatively slow resolution. Some software and websites tend to blur small images, which can make pixel art look bad. To avoid that, you are also given the option to resize on export, with a chosen interpolation method. This is useful when you want to share your art and make it easier for people to preview it. For most cases, sticking with the default Nearest interpolation should be fine.

You can choose which frames you want to save by changing the "Frame" option below the preview. The default setting is to include all frames, but you can set it to only include the currently selected frame, or select specific tags. Similarly, you can select which layers you want to include by changing the "Layers" option. You are also given the option to select the animation direction from either forward, backward or a ping-pong loop.

Export a sequence of images

By default, the export window is set to export a sequence of images, one for each frame, either as multiple files, or as a single animated file, if the file format is set to an animated image format, such as `.gif` or `.apng`.

If you save your animation as multiple files, then every file will have a number attached to its name. For example, if the name is `filename.png`, the exported file names will be `filename_0001.png`, `filename_0002.png`, `filename_0003.png` and so on. In the advanced options, you can find options to change the separator character(s) (in the above example, the character is the underscore (`_`), which is also the default). If these frames also have a frame tag, the name of the tag can be included in the file name as well, by enabling "Include frame tags in the file name". You can also choose to automatically create a new directory for each frame tag for extra organization, by enabling "Create new folder for each frame tag". This will place all of the frames of the same tag in a different directory.

Keep in mind that if you are working with a large canvas size and/or multiple frames, the `.gif` exporting process can be slow. Until we speed up the process, it is recommended to

export to `.apng` and use another tool, such as https://ezgif.com/apng-to-gif, to convert the `.apng` file to `.gif`.

Export a spritesheet

You can save the entire animation as a single image containing all of the frames. This is called a `spritesheet`. You can choose how many rows and columns you want your animation to be split into, in the spritesheet. Spritesheets can also be imported into Pixelorama.

The above project exported as a gif file with Pixelorama

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Palettes

Palettes are a way to organize colors in your projects.

Pixelorama ships on most platforms with several pre-made palettes. These palettes can be freely modified.

All palettes are physically stored on disk in Pixelorama data folder.

> **UNDO/REDO**
>
> Palettes currently don't support undo/redo so be careful with any modifications because they are permanent.

Palettes panel

The top of the panel consists of `Palettes selector`, `Edit palette button` and `Add new palette button`.

The main section is the `Palette grid`, displaying all colors present in the currently selected palette.

On the left of the `Palette grid` are `tools` used to edit the colors of the palette.

Creating a new palette

A new palette can be created by clicking `Add new palette button`.

Preset

You can select one of the presets to initialize a new palette with colors.

Preset	Description
Empty	Empty palette with no colors.
From Current Palette	Palette will be created with colors copied from the currently selected palette.
From Current Sprite	Colors will be imported from the currently opened image.
From Current Selection	Colors will be imported from the current selection.

Name

The name of the palette is also used as a palette file name.

Comment

The palette's description. Default Pixelorama palettes use this field to credit palette authors.

Width and Height

Palettes have set width and height (can be resized later) which define their grid size. Allowed values are from 1 to 64.

Create colors with alpha component

If this option is not toggled on, colors will be imported without alpha channel and transparency is ignored.

Get colors from

You can limit colors import to just some or all cels (as seen in the animation timeline).

Option	Description
Current frame	Colors imported from currently selected frame and all of its layers.
Selected cels	Colors imported only from selected cels.
All frames	Colors imported from all frames and their layers.

Editing palette metadata

Currently, selected palette metadata can be edited with `Edit palette button`.

The majority of properties are identical to create a new palette dialog.

> **PALETTE RESIZING**
>
> If the palette size is reduced positions of all colors will be reset to the beginning of the palette. If some colors don't fit in the new size they will be removed!

Name

If the palette name is changed the file name will also be changed accordingly.

Path

Displays where exactly in the file system the palette file is located.

Delete button

Permanently removes the palette from the file system.

Palette grid

Displays all colors present in the palette. Colors in the grid can be interacted with.

Interaction	Description
`Left click`	Color will be left selected and set as the left tool color.
`Right click`	Color will be right selected and set as the right tool color.
`Double click`	Double click on an existing color opens a color picker to change it's color.

> **PALETTE GRID PANNING**
> If palette size is greater than `8x8` it can be panned in the grid either with sliders or `Middle click + Mouse drag`.

Tools

Tools allow you to edit colors in the palette.

Button	Tool	Description
`+`	Add a new color	Left mouse click adds left tool color. Right mouse click adds right tool color.
`-`	Remove a selected color	Left mouse click removes left selected color. Right mouse click removes right selected color.

Colors movement

Colors can be moved and copied around in the palette grid.

Shortcut	Movement description
`Mouse drag`	Color switches with a color it is dropped on.
`Shift` + `Mouse drag`	Color is inserted where dropped. If dropped on another color the original color is moved to the right with every color next to it.
`Ctrl` + `Mouse drag`	Color is copied where dropped. If dropped on another color the original color is overwritten.

Palettes import

To import a palette, either `drag and drop` a palette file into Pixelorama window or use `File > Open` dialog.

An imported palette will be converted to `Pixelorama palette`, stored physically in the Pixelorama data folder and added to the palettes selection in `Palettes panel`.

The only exception to import behavior is `Image palette`. Importing an image to Pixelorama will open `Import Options` dialog in which you can select `Import as > New palette` to process an image to import a palette.

> **PALETTE EXTRACTION**
>
> You can import any image as a palette and extract colors from it. Every color from the image will be added to the palette only once. With bigger images, it can take longer time to finish.

Supported formats

Format	File extension
Pixelorama palette	.tres
GIMP GPL	.gpl
PAL	.pal

Format	File extension
Image palette	.png, .bmp, .hdr, .jpg, .jpeg, .svg, .tga, .webp
Pixelorama old palette	.json

FILE EXTENSIONS

Palette format detection uses file extensions to determine how to parse palette formats. An incorrect extension will result in an import error.

PRE-0.9 PIXELORAMA PALETTES

Palettes created in versions of Pixelorama prior to 0.9 have to be reimported. The save format was changed from `.json` to more Godot-compatible `.tres`.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Drawing

Drawing is the main feature of Pixelorama and it's what you will be doing most of the time. There are multiple tools that can be used for drawing in the canvas, with the main one being the **Pencil** tool.

Drawing Tools

You can use one of the available drawing tools in Pixelorama to bring your heart's content to life.

Name	Description	Default Shortcut
Pencil	Draw individual pixels on mouse position.	L: `P`, R: `Alt + P`
Bucket	Fill a same-color area or all of the pixels of the same color with a new color.	L: `B`, R: `Alt + B`
Line Tool	Create a straight line.	L: `L`, R: `Alt + L`
Rectangle Tool	Create a rectangle.	L: `S`, R: `Alt + S`
Ellipse Tool	Create an ellipse.	L: `C`, R: `Alt + C`

Tools that are not used directly for drawing but are very helpful for your drawing process:

Name	Description	Default Shortcut
Color Picker	Get the color of the pixel on mouse position.	L: `O`, R: `Alt + O`
Eraser	Erase individual pixels on mouse position.	L: `E`, R: `Alt + E`
Shading	See this page for a detailed explanation.	L: `U`, R: `Alt + U`

Pencil and Eraser

The main tools of Pixelorama. Used to draw or erase pixels at an individual pixel level. You can select from multiple brushes and change their size. You can also hold `Shift` before drawing to create lines between two points, and by also holding `Control`, you constrain them either by 15 degrees, or always making them be pixel-perfect angles by having the "Pixel Perfect" option enabled.

The eraser erases pixels by simply subtracting their color RGBA value by the given opacity value. If the opacity value is 255 then the color RGBA value becomes (0, 0, 0 ,0), which is equivalent to black that is completely transparent.

Pencil Tool Options

The Pencil tool also has an "Overwrite Color" tool option. If this is enabled, the color of the drawn pixels will always become the same color as the selected one. Otherwise, if the colors are transparent, their alpha values will blend. For example, if the color of the drawn pixels have are 30% transparent and your currently selected color is 25% transparent, the resulting transparency will be 55%. While if the overwrite color option is on, the resulting transparency will always be 25%.

The "Fill inside" option fills the entire area with the selected color between the first mouse-clicked position and the last, after the mouse button gets released.

Pixel Perfect

The Pencil and Eraser (and Shading!) tools have an option called "Pixel Perfect". This essentially creates "pixel perfect" lines for you by automatically removing the middle pixel of L-shapes. In other words, it removes extra adjacent pixels often referred to as "doubles" in order to create lines that are one-pixel thick.

Not pixel-perfect line on the left, compared to a pixel perfect line on the right.

Bucket

The bucket tool is a quick way to paint large sections of the image. You can select one of two fill areas. "Same color area" will fill the area around the mouse click position that is the same

color as the pixel on mouse position with the selected color, using a flood fill algorithm. "Same color pixels" will change all of the pixels of the selected cels of the same color as the one on mouse position with the selected color.

You can also choose to fill with patterns instead of a color.

Shape tools

The shape tools include the line tool, rectangle tool and ellipse tool. All of them are very straightforward to use; you click on where you want to place your shape, hold your mouse button and finally release it on where you want your shape to end. You can also change the shape's thickness, and for the rectangle and ellipse tools, you can choose whether to create hollow or filled with color shapes.

Similar to the selection tools, you can use keyboard shortcuts to modify how the shapes are being created.

- Mouse click + `Shift` = Create a 1:1 (square/circle) shape. For the line tool, `Shift` will restrain the angle to be pixel perfect, similar to how `Control` and `Shift` do for the Pencil tool.
- Mouse click + `Control` = Create a shape that expands from the center.
- Mouse click + `Alt` = Displaces the origin of the shape.

Color Picker

This one is fairly simple to comprehend. While not a drawing tool itself, the color picker tool is used for getting a sample of an existing color on the canvas with the purpose of replicating it, depending on which pixel the selection is done at.

Now, there's only one category available as an option for the time being, called "Pick for:". This means that the sample taken will replace one of the colors, either the **Left Color** or **Right Color**. Choosing **Left Color** replaces the left selected color, while right replaces the right selected color.

> **QUICK ACCESS**
>
> You can quickly access the color picker **without changing tools** by simply holding `Alt` and `Left/Right Click` the desired color to assign it to Left or Right tools respectively.

> **QUICK ACCESS**

> Another quick and helpful way to use the color picker is having it mapped to your **right mouse button** and having the **Left Color** option selected while having a drawing tool mapped to your **left mouse button.** The color picker can complement the drawing tool by selecting the color you next desire to paint with, making it an immediate way of switching between hues.

Mirroring

All drawing tools can work with symmetry. Symmetry in Pixelorama is either horizontal, vertical or both at the same time. It can be enabled from the tool options. If, for example, horizontal symmetry is enabled, whatever you draw will get mirrored, meaning it will also get drawn on the other horizontal side, having the same distance from the symmetry line as your original drawing. So if you draw a pixel two pixels from the left of the vertical symmetry line and horizontal symmetry is enabled, then the pixel two pixels from the right of the symmetry line and on the same vertical position will also get drawn. The same holds true when vertical symmetry is enabled.

If you have both horizontal and vertical mirroring enabled, then the drawn pixels will get repeated four times.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Selecting

You can select specific areas of your sprite to make changes at. When there is an available selection, any operation such as drawing, shading and image effect application will only affect the selected part(s) of the drawing. Selections also allow you to easily apply transformations to specific parts of the image, such as moving and scaling. An active selection is denoted by a marching ants effect around the selected pixels.

Selection Tools

You can use one of the available selection tools in Pixelorama. Each tool selects things slightly differently.

Name	Description	Default Shortcut
Rectangular Selection	Create a rectangular selection.	L: `R`, R: `Alt + R`
Elliptical Selection	Create an elliptical selection.	L: `Y`, R: `Alt + Y`
Polygonal Selection	Create a polygonal selection. Click where you want each point of the polygon to be. Double click to end your polygon and finalize your selection area.	L: `K`, R: `Alt + K`
Select By Color	Selects all parts of the cel that share the same color.	L: `W`, R: `Alt + W`
Magic Wand	Selects the neighboring same-color area of the pixel the cursor was in on mouse button press.	L: `Q`, R: `Alt + Q`
Lasso / Free Select Tool	Selects a rough outline of an area.	L: `F`, R: `Alt + F`

Similar to the shape drawing tools, you can use keyboard shortcuts to modify how the rectangular and elliptical selections are being created.

- Mouse click + `Shift` = Create a 1:1 (square/circle) selection.

- Mouse click + `Control` = Create a selection that expands from the center.
- Mouse click + `Alt` = Displaces the origin of the selection.

You can also easily change the selection's position and size from the tool options if you have a selection tool pressed. Keep in mind that if there is an active transformation, the content inside the selected parts will be affected as well. Otherwise, these options only affect the selection itself and not its content.

Modifying Selections

The selection tools also give you the ability to modify the already existing selection, by either adding to it, removing from it or creating intersections. By default, the selection tools replace the already existing selection with a new one. Keep in mind that you have to **press the keyboard modifiers first before the mouse button is pressed.**

- `Shift` + Mouse click = Add to selection.
- `Control` + Mouse click = Subtract from the selection.
- `Shift` + `Ctrl` + Mouse click = Create selection intersection.

Selecting All

You can easily select the entire canvas by either pressing `Control + A`, or by going to the Select menu and pressing **All**.

Clearing Selection

If you want to delete the entire selection, the easiest way is to deselect everything. You can do that by either pressing `Control + D`, or by going to the Select menu and pressing **Clear**. If you have the rectangular selection, elliptical selection, magic wand or lasso tools selected, clicking anywhere outside the canvas also clears the selection.

Inverting Selection

Inverting the selection means that all of the currently selected pixels will get deselected, while all of the unselected pixels will get selected. You can do this by either pressing `Control + I`, or going to the Select menu and pressing **Invert**. If you have everything selected, inverting will clear the selection. Likewise, if you have nothing selected, inverting will select everything.

Duplicating

If you **don't** have an active transformation, holding `Control + Alt + Mouse click` will *duplicate* whatever was below it and place it in active transformation.

> **DUPLICATING QUICKLY!**
>
> **During active transformation** `Control + Alt + Mouse click` will not only apply transformation but will also **retain** its content, which you can use again. Handy when duplicating something multiple times.

Copy, Paste, Cut and Delete

If you have an active selection, you can copy its contents by pressing `Control + C`. Then, you can paste it whenever you like, either on the same cel, on another cel or even a completely different project with `Control + V`. Pasting will create a new transformation which will be created at the same position as the original selection was when it was copied, but you will easily be able to move it to where you want. Keep in mind that copy/paste **does not** work at an Operating System level. Meaning, you won't be able to copy/paste image data between different applications on your device.

You can also easily delete the selected content with the `Delete` key. You can also cut with `Control + X`. Cut basically copies the selected content and then immediately deletes it.

Keep in mind that all of these operations only affect the currently selected cel and not the entire layer as of right now. You can also use the Edit menu to find these options instead of using keyboard shortcuts.

Selection Preferences

Under Edit -> Preferences -> Selection, you will find options to change the colors of the marching ants effect, as well as toggle its animation on or off.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Shading

Pixelorama's shading tool can be used to easily add lighting and shadows to the artwork. You can use it the same way you use the Pencil or Eraser tools, you mouse press on where the pixels of the sprite where you want to apply the shading. Similar to Pencil and Eraser, you can change the affected area size and brush. As of right now, there are two modes of shading, Simple Shading and Hue Shifting, which you can switch from the tool's options.

Simple Shading

This is a fairly straightforward method of shading. You can choose to either lighten or darken the area where you clicked, as well as the shading value. The bigger the number, the more shading will be applied. All it does is simply increase (with lighting) or decrease (with shading) the `Value` of the pixel's colors, which represents the color's brightness. It essentially just makes the same colors brighter or darker.

Hue Shifting

Hue Shifting is a bit more complex than Simple Shading. Instead of just shifting the `Value` of the color, we also shift their `Hue` and `Saturation` too. When we are brightening the colors, we shift their hue to move towards yellow (the limit is set to roughly 60 Hue), decrease their saturation and increase their value. On the contrary, when we are darkening the colors, their hue shifts towards blue (the limit is set to roughly 240 Hue), their saturation increases and their value decreases. This is all done automatically by the Shading tool, but you can also configure the amount the Hue, Saturation and Value change from the tool options.

Here is a video by Brandon James Greer that explains the concept of hue shifting.

Hue Shifting in Pixel Art (Color Tutorial)

HUE SHIFTING LIMITS

The hue shifting mode does not always change the hue, saturation and value of the colors exactly as much as the values have been set by the user. For example, a color with a hue of 70 and the shading tool set to hue shifting, lighten and a hue of, let's say, 20, will not result in a color with a hue of 50 (because 70 - 20 = 50), but it will rather stay limited to 60, as this is the hard limit of lighting. Similarly, the hard hue limit of darkening is 240. If you want to exactly change the HSV values of a color, you can use the `Adjust Hue/Saturation/Value` image effect under the Image menu.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Tools

Tools are the main way to interact with your artwork.

Pixelorama's toolbar, located on the left side of the application window.

Left and right mouse buttons

> **TOOLS FOR LEFT AND RIGHT MOUSE BUTTONS**
>
> Left-clicking a tool will assign this tool to your left mouse button and right-clicking it will assign the tool you clicked to your right mouse button. If you left-click on the canvas, your left tool will be activated, and if you right-click on the canvas, your right tool will be activated.

In Pixelorama, you are given the ability to map a different tool to each mouse button, left and right. This essentially means that you can use a different tool with your left mouse button, and a different tool with your right mouse button. The two mouse buttons can be used **independently** of one another. In the toolbar, you will see one of the tool buttons having a half background on the left side of the button as **blue**. This represents the left

mouse button, and it means that this specific tool has been **mapped to the left mouse button**. Similarly, the tool button with that has has a half background on the right side as **orange is mapped to the right mouse button**. If a tool button has both a blue and orange background, then this means that this tool has been mapped to both left and right mouse buttons.

So, to sum up, blue represents the left mouse button and orange represents the right mouse button. To map a tool to your left mouse button you can left-click the button of the tool you want to map in the toolbar, and to map a tool to your right mouse button you can right-click the tool button.

When you are on your canvas, if you left-click, the tool that you left-clicked on the toolbar will be used and if you right-click, the tool that you right-clicked will be used.

A screenshot example of the Pencil tool having a blue half background, which is mapped to the left mouse button and the Eraser tool having an orange half background, which is mapped to the right mouse button.

The tools

These are all of the available tools in Pixelorama in order, from top to bottom.

The selection tools

Name	Description	Default Shortcut
Rectangular Selection	Create a rectangular selection.	L: `R`, R: `Alt + R`
Elliptical Selection	Create an elliptical selection.	L: `Y`, R: `Alt + Y`
Polygonal Selection	Create a polygonal selection. Click where you want each point of the polygon to be. Double click to end your polygon and finalize your selection area.	L: `K`, R: `Alt + K`

Name	Description	Default Shortcut
Select By Color	Selects all parts of the cel that share the same color.	L: `W`, R: `Alt + W`
Magic Wand	Selects the neighboring same-color area of the pixel the cursor was in on mouse button press.	L: `Q`, R: `Alt + Q`
Lasso / Free Select Tool	Selects a rough outline of an area.	L: `F`, R: `Alt + F`
Move	Moves the content of the active cel. Not a selection tool, see transforming.	L: `T`, R: `Alt + T`

Canvas navigation tools

Name	Description	Default Shortcut
Zoom	Allows the user to change the zoom level of the canvas.	L: `Z`, R: `Alt + Z`
Pan	Allows the user to move around the canvas.	L: `M`, R: `Alt + M`

Drawing tools and drawing-assisting tools

Name	Description	Default Shortcut
Color Picker	Get the color of the pixel on mouse position.	L: `O`, R: `Alt + O`
Pencil	Draw individual pixels on mouse position.	L: `P`, R: `Alt + P`
Eraser	Erase individual pixels on mouse position.	L: `E`, R: `Alt + E`
Bucket	Fill a same-color area or all of the pixels of the same color with a new color.	L: `B`, R: `Alt + B`

Name	Description	Default Shortcut
Shading	See this page for a detailed explanation.	L: `U`, R: `Alt + U`
Line Tool	Create a straight line.	L: `L`, R: `Alt + L`
Rectangle Tool	Create a rectangle.	L: `S`, R: `Alt + S`
Ellipse Tool	Create an ellipse.	L: `C`, R: `Alt + C`

Hovering your mouse over a tool button will reveal a small display that can give you more information about the tool.

Tool options

Almost every tool has certain properties that the user can tweak, such as brush type and size, different modes, color or mirroring. The tool options are located in the right side of the application window and they are separated in left tool and right tool settings. These, of course, correspond to the two tools you have mapped at your two mouse buttons. The two colors are also mapped to different mouse buttons. The left one is for the left mouse button tool, and the right one is for the right mouse button tool.

> **CHANGE MODE QUICKLY!**
>
> Most of the tools have some binary property that has to be changed often. Holding `Ctrl` will allow you to quickly alternate between the properties for a given tool as follows:

Tool	Alternating Property
Zoom	Zoom in / Zoom out
Color picker	Pick for left color / Pick for right
Pencil	Overwrite color on / off
Bucket	Fill same color area / Fill same color pixels
Shading	Lighten / Darken option

The tool options change every time the user changes the available tool, and most tools have their own unique options.

An example of tool options, for the Pencil and Eraser tool.

Changing colors

On the above screenshot, on the top side, you can see two color buttons, for the left and right tools. To change them, you can either click on them and a color picker window will appear, or you can also choose a color from the palette. Left-clicking a swatch from the palette will change the left color, and right-clicking a swatch will change the right color. You can also switch between them by clicking on the double-arrow button above the two color buttons (also mapped to the `X` keyboard button by default). The small button below the two color buttons is used to change them back to the default colors, which are black for the left color, and white for the right color.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Transforming

Currently, there are three types of transformations in Pixelorama. Translation (moving), scale and rotation. Transformations can either affect the currently active entire cels, or selections.

Keep in mind that **any content moved outside the canvas boundaries will be lost**.

Move

To move pixels around in the canvas, you can use the Move tool. It works by simply holding the mapped mouse button and dragging the cursor to where you want to move the content. If there is no active selection, it will move the entire content of the active cels. If there is a selection, it will start a selection transformation, and it will move both the selection itself and its content. To move a selection, it is also possible to use a selection tool and click inside the selected area and, while the mapped mouse button is being held, drag the cursor, just like you do with the Move tool. There are some keyboard shortcuts that can help you with movement. **The order of the key press and mouse button press matters for the selection tools.**

- Mouse movement + `Shift` = Moves & snaps selection to an axis.
- Mouse movement + `Control` = Moves & snaps selection to the rectangular grid.
- `Alt` + Mouse movement = Moves selection without content. **This only works for the selection tools and not the Move tool.**
- `Control` + `Alt` + mouse movement = Copies selection and moves it without cutting it from the original position. This is basically a quick way to copy and move stuff around without `Control + C` and `Control + V`. **This only works for the selection tools and not the Move tool.**

If you have a selection tool active, you can also use the arrow keys to move the selection and contents. Pressing `Alt` only moves the selection without content, and holding `Control` moves the selection by the size of the rectangular grid tiles. You can also manually change the position of the selection from the tool options. If there is an active transformation, the content will be affected as well, otherwise, it will only move the selection itself without its content.

Scale

To resize your image, you can go to the Image menu and select **Scale Image**. There you can choose the new size of your entire project. You can also just expand the canvas without

affecting the content that is inside by going to the Image menu and pressing **Resize Canvas**. Furthermore, you can go to Image -> Crop Image, which will remove all of the transparent pixels that surround your drawing, by shrinking the size of the image. Of course, all of these options will affect all of the cels of the project.

If you want you resize only selected part(s) of your artwork, you can create a selection and use one of the eight gizmos that appear on each side of the selection borders. To use them, you can simply left-click on them, hold, then move your cursor. Moving it outwards will expand the size, while moving it inwards will shrink it. Using gizmos will start a selection transformation. Like with movement, there are some keyboard shortcuts you can use to give you more control over the transformation.

- Mouse click + `Shift` = Keeps the original aspect ratio of the selection.
- Mouse click + `Control` = Expands from the center.
- `Alt` + Mouse click = Only affects the selection and not the content.

You can also manually change the size of the selection from the tool options. If there is an active transformation, the content will be affected as well, otherwise, it will only scale the selection itself without its content.

Rotate

To rotate your image, you can go to the Image menu and select Rotate Image. There, you can choose one of the three rotation algorithms. You can experiment with every one of them to see what works best for you. You are also given the option to only affect the selected part of the image or the entire area of the active cels. In the future, there will most likely be gizmos that rotate the selection directly, just like there are for scaling.

Having An Active Transformation

When moving or scaling a selection with its contents, the selected pixels will suddenly become transparent. This indicates that a selection transformation has started. This means that your changes haven't yet been applied to the image. When there is an active transformation, moving and scaling will only affect the transformed content and nothing else, until it is applied.

Applying A Transformation

To apply your changes, you can either press `Enter`, or by doing any other action, such as drawing, applying an image effect, selecting another cel, adding a new frame or layer, etc.

Doing any of these will automatically apply your changes.

Canceling A Transformation

To cancel a transformation, you can either press `Escape` or undo. You will not be able to redo your actions if you decide to cancel.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Brush

Brushes can be used to easily draw complex shapes and patterns, without having to copy and paste them. The user can also change their color and size.

To access brushes, select the pencil tool and click on the white square.

Then, the different brushes are shown.

Brush Types

Pixelorama comes with three default brushes, the usual pixel/square brush, a hollow circle and a filled circle brush. These brushes are based on algorithms, while all of the other brushes as of right now are based on image data. The main difference is that the image-based brushes are scaled the same way that images are being scaled, which may not always have the result we desire. The image-based brushes are also divided into two types, File

brushes and Project brushes. File brushes are stored as external `.png` files, while project brushes are stored in the project itself. File brushes can also be Random brushes. Each time you draw using a random brush, the result will be something different!

Image-based brushes will also have one extra option in the tool settings; a "Brush Color From" slider which goes from 0 to 100. If it's set to 0, the resulting color(s) of the brush will come entirely from the image data themselves. Otherwise, if set to 100, the brush color will come from the currently selected color. This setting is important to have because, since image-based brushes rely on image data, this means that they also get to keep the original colors of the image data.

Making Your Own Custom Brushes

File Brushes

Pixelorama gives you the option to create your own custom brushes, which you can later use to draw. There are two brush types, File and Project brushes. File brushes are `.png` files saved in the `pixelorama_data/Brushes` directory. They are available for all projects. To create a "File" custom brush, all you need to do is create your brush and save it as a `.png` file in the `pixelorama_data/Brushes` directory. They will be found on the "File" tab.

Random Brushes

To make a random brush, save your sprites as multiple `.png` files, with `"~"` at the beginning of their filename, and save them in the same sub-directory. For example, if you want to save a grass random brush, you can create a `pixelorama_data/Brushes/Grass` directory, and name the files something like `~Grass1.png`, `~Grass2.png`, `~Grass3.png`, `~Grass4.png`, and so on. The more files you add, the more variation (random values) your random brush will have.

Project Brushes

Project brushes are project-dependent, which means that they're not available for all projects. They are saved inside `.pxo` files. To create a Project brush you can select the part of the drawing you want to save as brush, and press `Control + B` or go to Edit, New Brush. They will be found on the "Project" tab.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Cel

In Pixelorama, the term "Cel" (from celluloid) refers to the intersection of frames and layers of the project. Basically, every frame in a project has as many cels as there are layers. You could think of frames as a collection of cels, for every layer. So for example, if the project has 3 frames and 4 layers, each frame would have 4 cels, making the total cels 3 x 4 = 12.

Cels can be found in the timeline. They are the square buttons you can click to edit. You can select/deselect multiple by holding the `Control` key and left-clicking individual cels, or hold the `Shift` key and left-click a cel to also select all of the cels between the one that was clicked and the one that has been selected last.

Linked cels

You can also right-click cels to link them together. Linked cels share the same image data, meaning that if you make changes to one linked cel, the changes are shared with all other linked cels. Keep in mind that, as of right now, linked cels are layer-local. You cannot have cels linked that belong to different layers, only different frames. Linked cels are marked with a green outline.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Project

In Pixelorama, the term "Project" refers to the entire sprite, which contains all of the frames it might have. Each project can be saved as a `.pxo` file. Multiple projects can be opened in Pixelorama at the same time, with each one having its own tab.

A Project contains the following properties:

- A set of frames for animations, along with their duration, animation tags and FPS, as well as multiple layers. Frames and layers intersect and create cels, which are the editable images themselves of the project.
- A size - width and height - in pixels.
- A name.
- Brushes that are local to that specific project.
- Guides - lines that help you draw.

Saving a project

See this page to learn how to save a project.

.pxo files

Pxo files are Pixelorama's custom file format that saves the entire content of the project. It cannot be opened in other image editors.

File structure

The file essentially consists of two lines. The first line is all of the project's metadata, such as project name, size, number of layers, number of frames, tags, project brushes, fps etc. They are stored in JSON form, so they can easily be read by other software.

The second line contains all of the image data as buffers. First, all of the image data for every cel is being stored. It starts from the very first cel, which is equivalent to the bottom-most layer and the first frame, then it continues for every layer from bottom to top and once it reaches the final layer, moves on to the next frame, where it again starts from the bottommost layer and continues saving up to the top. After all of the cel image data has been saved, the image data of the project brushes are saved if the project has any.

So, in order for a software to read the data of a pxo, it first reads the first line as a JSON object, and then it loops through the buffers, which all have the same size as the project size, until it reads all the frames and layers. After that, if there are more buffers, they correspond to the project brushes, the size of which is stored in the JSON object.

ZSTD compression

During save, you are given the option to select ZSTD compression. If you do, this will reduce your pxo size, but if your project is large and has a lot of content in it, it may take some time to save and load it. Saving projects with ZSTD compression is currently unsupported in the Web version. If you are writing software that supports .pxo files, make sure to detect if the files are compressed first.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Extension Basics

Making an Extension

Since **Pixelorama v0.10**, you can add/remove extensions to/from Pixelorama. Extensions are distributed as PCK (.pck) or ZIP (.zip) files and you can manage them from **Edit > Preferences > Extensions**. This is a step-by-step guide of how you can make an extension for pixelorama. For this tutorial the extension we'll make will be called "Example"

Prerequisites

First of all, there are a few prerequisites for you to start creating extensions.

- Some knowledge of GdScript.
- A ready to use Godot executable (See the Pixelorama Channgelog to figure out which version to use).
- Source code of Pixelorama (This is optional but will speed up your debugging workflow).

Process

1. First of all make a new Godot project of name "Example" (the project name is irrelevant but for simplicity we'll name it the same as our extension).
2. After that make a directory structure as shown in the image below.

```
res://
 └ src
    └ Extensions
       └ Example                    <----- This needs to be of the same name as the extension
          ├ extension.json
          ├ Main.tscn
          └ Main.gd
```

The File Structure of any basic extension.

In the "Example" folder you can see three files;

- extension.json
- Main.tscn
- Main.gd

These files are the bare minimum required for an extension to function properly. Let's go through each of them one by one:

extension.json

The `extension.json` file serves as a basic configuration file for the extension. A proper `extension.json` file should contain the below content

```
{
    "name": "Example",
    "display_name": "Anything you want",
    "description": "What the extension does",
    "author": "Your Name",
    "version": "0.1",
        "supported_api_versions": [3],
    "license": "MIT",
    "nodes": [
        "Main.tscn"
    ]
}
```

Key	Description
name	The extension name (in this case Example)
display_name	The name actually shown by pixelorama in it's extension list (it can be anything)
author	Author of the extension
version	version of the extension, it is a floating value so versions like 1.0.3 etc. are invalid
supported_api_versions	(Optional) Announces the Api level of the extension
license	Name of license used by the extension
nodes	Array containing name of scenes that should be instanced as soon extension gets enabled (in this case it is `Main.tscn`)

API VERSIONS

> `supported_api_versions` is an optional key that is used by pixelorama to determine if it is compatible with it's Api (E.g `[ 3 ]` means you are making extension for Pixelorama 0.11.x). If your extension is compatible with multiple api e.g (level 2 and 3) then use `[ 2, 3 ]`. To find the Api level of pixelorama open the `project.godot` file in **Pixelorama's Source** and look for `config/ExtensionsAPI_Version`

Main.tscn

As mentioned above, the `Main.tscn` will be instanced in pixelorama as soon as the extension gets enabled. a GdScript file (In this case `Main.gd`) is attached to this scene.

Main.gd

As mentioned above, the `Main.gd` is a script written in GdScript, that will be run as soon as the extension gets enabled. It can be used to

- Instance further scenes (.tscn files).
- Or contain the core extension code.

The content of a basic `Main.gd` file is;

```
extends Node

# NOTE: use get_node_or_null("/root/ExtensionsApi") to access the extension api.

# Runs as soon as extension is enabled. This script can act as a setup for the
extension.
func _enter_tree() -> void:
        pass


func _exit_tree() -> void:  # Extension is being uninstalled or disabled
        # remember to remove things that you added using this extension
        pass
```

Limitations and Workarounds

1. New custom classes (e.g `class_name Foo`) can't be created in the extension, but there is a workaround. Don't assign scripts custom class names and instead of doing `Foo.new()` if say your script "*Foo.gd*" is located at `res://src/Extensions/Example/CustomClasses/Foo.gd` then you can use

```
preload(res://src/Extensions/Example/CustomClasses/Foo.gd).new()
```

2. If your extension needs to use pixelorama's classes (e.g `Project`, `BaseCel`, `BaseTool` etc) make empty class scripts and place them in `res://src/Extensions/Example/EmptyClasses/`. For example an empty class scripts of `Project` class will look like;

```
# This is example of an empty Project class
class_name Project
extends Reference

# Empty Variables (variable containing only name and type but no value)
# obviously they are not all of Project class variables
# They are written to remove (Variable not found in class: Project) errors
# (add more variables from the real Project class if your extension needs them)
var name: String
var size: Vector2
var undo_redo: UndoRedo
var tiles: Tiles
var undos: int
var can_undo: bool
var fill_color: Color
var has_changed: bool


# Empty functions (they don't contain any actual code)
# obviously they are not all of Project class functions,
# They are written to remove (Method not found in class: Project) errors
# (add more functions below from the real Project class if your extension needs
them)
func _init(_frames := [], _name := tr("untitled"), _size := Vector2(64, 64)) ->
void:
        return  # As the functions returns void, we are using return without a
value


func remove() -> void:
        return  # As the functions returns void, we are using return without a
value


func serialize() -> Dictionary:
        # As the functions returns a value, we are returning the most basic value
of the same type.
        # for a Dictionary, it's smallest value is just an empty dictionary
        return {}
```

Exporting the Extension

Now that you have the basic extension code ready, let's export it;

1. From the top bar in Godot editor go to **Project > Export** and choose any platform option (the extension should work on other platforms regardless of which platform you choose).

2. In then go to the **Resources** tab and do the following changes;

   - `Export Mode` should be set to `Export all resources in the project`
   - `Filters to export non-resource files/folders` should be set to `*.json`
   - `Filters to exclude files/folders from project` should be set to `res://src/Extensions/Example/EmptyClasses/*`

Extension Export Options.

3. After that, press `Export PCK/Zip` and export it as a PCK file (both Zip and PCK extensions are recognized by pixelorama but PCK is recommended). The name of exported pck should be the same as the `name` of your extension, in this case it should be `Example.pck`

Installing the Extension

To install an extension, from pixelorama's top menu go to **Edit > Preferences > Extensions** and click `Add Extension`.

> **DRAG AND DROP**

A more easier way would be to drag and drop the extension in Pixelorama.

Adding an Extension.

After it has appeared in the list of extensions, select it press `Enable`.

Debugging an Extension

In order to debug an extension, you need to have the source-code of your target version of Pixelorama open in another godot instance. To start debugging, **run pixelorama from godot** and follow the steps in Installing the Extension. The debugger of Godot instance housing pixelorama's source-code will be used to debug the extension.

Debugging of an extension (stack trace shown as a result of non existent function).

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

ExtensionsApi

> **0.X API**
>
> This is the Api documentation of Pixelorama 0.x, After Pixelorama 1.0 instead of showing documentation, this page will show you how to get it through godot documentation comments in pixelorama's source-code.

Description

This Api gives you the essentials to develop a working extension for Pixelorama. The Api consists of many smaller Apis, each giving access to different areas of the Software;

To access this anywhere in the extension use `get_node_or_null("/root/ExtensionsApi")`

> **TIP**
>
> Keep in mind that this API is targeted towards users who are not fully familiar with Pixelorama's source code. If you need to do something more complicated and more low-level, you would need to interact directly with the source code.

Properties

- general
- menu
- dialog
- panel
- theme
- tools
- selection
- project
- exports
- signals

Method Descriptions

- `int` `get_api_version()`

Returns the version of the ExtensionsApi.

GeneralAPI

Gives Access to the general stuff.

Description

This part of Api provides stuff like commonly used Autoloads, App's version info etc the most basic (but important) stuff.

Method Descriptions

● **String** `get_pixelorama_version()`

Returns the current version of pixelorama.

● **ConfigFile** `get_config_file()`

Returns the `ConfigFile` contains all the settings (Brushes, sizes, preferences, etc...).

● **"src/Autoload/Global.gd"** `get_global()`

Returns the Global autoload used by Pixelorama. Contains references to almost all UI Elements, Variables that indicate different settings etc..., In short it is the most important autoload of Pixelorama.

● **"src/Autoload/DrawingAlgos.gd"** `get_drawing_algos()`

Returns the DrawingAlgos autoload, contains different drawing algorithms used by Pixelorama.

● **ShaderImageEffect** `get_shader_image_effect()`

Gives you a new ShaderImageEffect class. this class can apply shader to an image. It contains method: `generate_image(img: Image, shader: Shader, params: Dictionary, size: Vector2)` Whose parameters are identified as:

Parameter	Description
img	Image that the shader will be pasted to (Empty Image of size same as project).

Parameter	Description
shader	preload of the shader.
params	A dictionary of params used by the shader.
size	It is the project's size.

● **Node** `get_extensions_node()`

Returns parent of the nodes listed in extension.json -> "nodes".

● **Canvas** `get_canvas()`

Returns the main `Canvas` node, normally used to add a custom preview to the canvas.

MenuAPI

Gives ability to add/remove items from menus in the top bar.

Enumerations

enum `@unnamed_enums`:

- FILE = 0
- EDIT = 1
- SELECT = 2
- IMAGE = 3
- VIEW = 4
- WINDOW = 5
- HELP = 6

Method Descriptions

● **int** `add_menu_item(menu_type: int, item_name: String, item_metadata: Variant, item_id := -1)`

Adds a menu item of title `item_name` to the `menu_type` defined by `@unnamed_enums`. `item_metadata` is usually a window node you want to appear when you click the `item_name`. That window node should also have a `menu_item_clicked` function inside its script. Index of the added item is returned (which can be used to remove menu item later on).

- **void** `remove_menu_item(menu_type: int, item_idx: int)`

Removes a menu item at index `item_idx` from the `menu_type` defined by `@unnamed_enums`.

DialogAPI

Gives access to common dialog related functions.

Method Descriptions

- **void** `show_error(text: String)`

Shows an alert dialog with the given `text`. Useful for displaying messages like "Incompatible API" etc...

- **Node** `get_dialogs_parent_node()`

Returns the node that is the parent of dialogs used in pixelorama.

- **void** `dialog_open(open: bool)`

Tells pixelorama that some dialog is about to open or close.

PanelAPI

Gives access to Tabs and Dockable Container related functions.

Method Descriptions

- **void** `set_tabs_visible(visible: bool)`

Sets the visibility of dockable tabs.

- **bool** `get_tabs_visible()`

Gets the visibility of dockable tabs.

- **void** `add_node_as_tab(node: Node)`

Adds the `node` as a tab. Initially it's placed on the same panel as the tools tab, but can be changed through adding custom layouts.

- **void** `remove_node_from_tab(node: Node)`

Removes the `node` from the DockableContainer.

ThemeAPI

Gives access to theme related functions.

Method Descriptions

● **void** `add_theme(theme: Theme)`

Adds the `theme` to **Edit > Preferences > Interface > Themes**.

● **int** `find_theme_index(theme: Theme)`

Returns index of the `theme` in preferences.

● **Theme** `get_theme()`

Returns the current theme resource.

● **bool** `set_theme(idx: int)`

Sets a theme located at a given `idx` in preferences. If theme set successfully then return true, else false.

● **void** `remove_theme(theme: Theme)`

Remove the `theme` from preferences.

ToolAPI

Gives ability to add/remove tools.

Method Descriptions

● **void** `add_tool(tool_name: String, display_name: String, shortcut: String, scene: PackedScene, extra_hint := "", extra_shortucts := [], layer_types: PoolIntArray = [])`

Adds a tool to pixelorama with name `tool_name` (without spaces), display name `display_name`, tool scene `scene`, layers that the tool works on `layer_types`, `extra_hint` (text that appears when mouse havers tool icon), primary shortcut name `shortcut` and any extra shortcuts `extra_shortucts`.

At the moment extensions can't make their own shortcuts so you can ignore shortcut and extra_shortucts.

● **void** `remove_tool(tool_name: String)`

Removes a tool with name `tool_name` and assign Pencil as left tool, Eraser as right tool.

SelectionAPI

Gives access to pixelorama's selection system.

Method Descriptions

● **void** `clear_selection()`

Clears the selection Gizmo.

● **void** `select_all()`

Select the entire region of current cel.

● **void** `select_rect(rect: Rect2, operation: int)`

Selects a portion defined by `rect` of the current cel. `operation` influences it's behaviour with previous selection rects (0 for adding, 1 for subtracting, 2 for intersection).

● **void** `move_selection(destination: Vector2, with_content: bool, transform_standby: bool)`

Moves a selection to `destination`, with content if `with_content` is true. If `transform_standby` is true then the transformation will not be applied immediatelyunless `Enter` is pressed.

● **void** `resize_selection(new_size: Vector2, with_content: bool, transform_standby: bool)`

Resizes the selection to `new_size`, with content if `with_content` is `true`. If `transform_standby` is `true` then the transformation will not be applied immediately unless `Enter` is pressed.

● **void** `invert()`

Inverts the selection gizmo.

● **void** `make_brush()`

Makes a project brush out of the current selection's content.

- **void** `copy()`

Copies the selection content (works in or between pixelorama instances only).

- **void** `paste(in_place: bool)`

Pastes the selection content.

- **void** `delete_content()`

Deletes the drawing on current cel enclosed within the selection's area.

ProjectAPI

Gives access to basic project manipulation functions.

Method Descriptions

- **Project** `new_project(frames: Array, name: String, size: Vector2, fill_color: Color)`

Creates a new project (with new tab) with name `name`, size `size`, fill color `fill_color` and frames `frames`. The created project also gets returned.

`frames` is an **Array** of type **Frames**. Usually it can be left as [].

- **void** `switch_to(project: Project)`

Switches to the tab that contains the `project`.

- **Project** `get_current_project()`

Returns the project in focus.

- **Dictionary** `get_project_info(project: Project)`

Returns a dictionary containing all the project information.

- **BaseCel** `get_current_cel()`

Returns the current cel. Cel type can be checked using function `get_class_name()` located inside the cel. It's type can be **GroupCel**, **PixelCel**, **Cel3D**, or **BaseCel**.

- **BaseCel** `get_cel_at(project: Project, frame: int, layer: int)`

Frames are counted from left to right, layers are counted from bottom to top. Frames/layers start at "0" and end at `project.frames.size() - 1` and `project.layers.size() - 1` respectively.

● **void** `set_pixelcel_image(image: Image, frame: int, layer: int)`

Sets an `image` at `frame` and `layer` on the current project. Frames are counted from left to right, layers are counted from bottom to top.

● **void** `add_new_frame(after_frame: int)`

Adds a new frame in the current project after frame `after_frame`.

● **void** `add_new_layer(above_layer: int, name: String, type: res://src/Autoload/Global.gd.LayerTypes)`

Adds a new Layer of type `type` with name `name` in the current project above layer `above_layer` (`above_layer` = 0 is the bottom-most layer and so on).

`type`	class
0	PixelLayer
1	GroupLayer
2	3DLayer

ExportAPI

Gives access to adding custom exporters.

Enumerations

enum `ExportTab`:

- IMAGE = 0
- SPRITESHEET = 1

Method Descriptions

● **int** `add_export_option(format_info: Dictionary, exporter_generator: Object, tab: ExportTab, is_animated: bool)`

`format_info` has keys: `extension` and `description` whose values are of type **String** e.g:

`format_info = {"extension": ".gif", "description": "GIF Image"}`

`exporter_generator` is a node with a script containing the method `override_export()` which takes 1 argument of type **Dictionary** which is automatically passed to `override_export()` at time of export and contains keys: `processed_images`, `durations`, `export_dialog`, `export_paths`, `project`

If the value of `tab` is not in `ExportTab` then the format will be added to both tabs.

Returns the index of exporter, which can be used to remove exporter later.

● **void** `remove_export_option(id: int)`

Removes the exporter with `id` from Pixelorama.

SignalsAPI

Gives access to the basic commonly used signals.

Description

Gives access to the basic commonly used signals. Some less common signals are not mentioned in Api but could be accessed through source directly.

Method Descriptions

● **void** `connect_project_changed(target: Object, method: String)`

connects a signal to method present in target, that emits whenever you switch to some other project.

● **void** `disconnect_project_changed(target: Object, method: String)`

reverse of `connect_project_changed()`.

● **void** `connect_cel_changed(target: Object, method: String)`

connects a signal to method present in target, that emits whenever you select a different cel.

- **void** `disconnect_cel_changed(target: Object, method: String)`

reverse of `connect_cel_changed()`.

- **void** `connect_tool_color_changed(target: Object, method: String)`

connects a signal to method present in target, that emits whenever a tool changes color.

- **void** `disconnect_tool_color_changed(target: Object, method: String)`

reverse of `connect_tool_color_changed()`.

- **void** `connect_current_cel_texture_changed(target: Object, method: String)`

connects a signal to method present in target, that emits whenever texture of the currently focused cel changes.

- **void** `disconnect_current_cel_texture_changed(target: Object, method: String)`

reverse of `connect_current_cel_texture_changed()`.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Examples

> ⚠ **CODE VERSION**
>
> The examples below assume you are developing for the latest Pixelorama release

Below are some example codes for different basic types of extensions. To use them, create an extension and paste your desired code from here to `Main.gd`;

- Bare Minimum
- Add new Tab
- Add Menu Item
- Change Font
- Project Manipulation
- Custom Exporter

Bare Minimum

The most basic code.

```
extends Node


# This script acts as a setup for the extension
func _enter_tree() -> void:
	pass


func _exit_tree() -> void:  # Extension is being uninstalled or disabled
	# remember to remove things that you added using this extension
	pass
```

Add new Tab

Adds a simple Panel as a tab, which will be placed at the same place as the Tools tab by default.

```
extends Node

onready var extension_api: Node  # Variable for keeping reference to the Api
```

```
# some references to nodes that will be created later
var panel


# This script acts as a setup for the extension
func _enter_tree() -> void:
	extension_api = get_node_or_null("/root/ExtensionsApi")  # Accessing the
Api

	# add a test panel as a tab  (this is an example) the tab is located at the
same
	# place as the (Tools tab) by default
	panel = Panel.new()
	panel.name = "This is a new panel"  # This is optional
	extension_api.panel.add_node_as_tab(panel)


func _exit_tree() -> void:  # Extension is being uninstalled or disabled
	# remember to remove things that you added using this extension
	extension_api.panel.remove_node_from_tab(panel)
```

Add Menu Item

Adds a menu item in Help menu, which displays a message when clicked on it.

```
extends Node

onready var extension_api: Node  # Variable for keeping reference to the Api

var item_id: int

# This script acts as a setup for the extension
func _enter_tree() -> void:
	extension_api = get_node_or_null("/root/ExtensionsApi")  # Accessing the
Api
	var type = extension_api.menu.HELP

	item_id = extension_api.menu.add_menu_item(type, "Show Message", self)
	# the 3rd argument (in this case "self") will try to call
"menu_item_clicked"
	# (if it is present)


func menu_item_clicked():
	# Do some stuff
	extension_api.dialog.show_error("You Tickled Me :)")


func _exit_tree() -> void:  # Extension is being uninstalled or disabled
```

```
        # remember to remove things that you added using this extension
        extension_api.menu.remove_menu_item(extension_api.menu.HELP, item_id)
```

Add a Theme

Adds a theme to pixelorama **Preferences > Interface > Themes** (you need to have a theme resource beforehand and that theme resource must derive from one of the main pixelorama themes)

```
extends Node

onready var extension_api: Node  # Variable for keeping reference to the Api

var theme = load("res://path_to_some_theme.tres") # Replace this with your theme
resource
# your theme resource must be a derivative of themes from
# https://github.com/Orama-Interactive/Pixelorama/tree/master/assets/themes
# or else a CRASH might occur


# if a theme from the extension was set in preferences, then it will be
automatically be
# set when pixelorama is launched again
func _enter_tree() -> void:
        extension_api = get_node_or_null("/root/ExtensionsApi")  # Accessing the
Api
        if theme:
                extension_api.theme.add_theme(theme)  # Adds the theme to
preferences


func _exit_tree() -> void:  # Extension is being uninstalled or disabled
        if theme:
                extension_api.theme.remove_theme(theme)  # Adds the theme to
preferences
```

Change Font

Sometimes you only need a different font instead of a different theme. This is much easier to achieve than adding themes.(you need to have a `.ttf` font resource beforehand)

```
extends Node
# used If your goal is to only use a different font

onready var extension_api: Node  # Variable for keeping reference to the Api

var font_res = load("res://path_to_some_font.ttf")  # Replace this with your font
```

```
resource
var old_font

# This script acts as a setup for the extension
func _enter_tree() -> void:
        extension_api = get_node_or_null("/root/ExtensionsApi")  # Accessing the
Api
        if font_res:
                old_font =
extension_api.general.get_global().control.theme.default_font.font_data
                #set the new font

extension_api.general.get_global().control.theme.default_font.font_data = font_res


func _exit_tree() -> void:
        if font_res and old_font:
                #set the default font back again

extension_api.general.get_global().control.theme.default_font.font_data = old_font
```

Project Manipulation

Wish to automate some project behaviour? this will demonstrate some of the basic functions provided by the project api. This example adds a menu button to the edit menu and on clicking it some project manipulations (instructed in `menu_item_clicked()`) will get carried out.

```
extends Node

# I will show you some common stuff to manipulate projects
# (I will make use of the menu api as well)

# To know the available methods of (timeline) see:
# https://github.com/Orama-
Interactive/Pixelorama/blob/master/src/UI/Timeline/AnimationTimeline.gd

# NOTE: some more advanced methods can be accessed from timeline node (un-comment
line below to get them)
#onready var timeline = ExtensionsApi.general.get_global().animation_timeline

onready var extension_api: Node  # Variable for keeping reference to the Api

var item_id: int
var type: int


func _enter_tree() -> void:
```

```
        extension_api = get_node_or_null("/root/ExtensionsApi")  # Accessing the
Api
        type = extension_api.menu.EDIT
        item_id = extension_api.menu.add_menu_item(type, "Click Me 4 times", self)


func _exit_tree() -> void:  # Extension is being uninstalled or disabled
        # remember to remove things that you added using this extension
        extension_api.menu.remove_menu_item(type, item_id)


################## Test Project methods whenever we click the menu button
############
var thing_to_do := 0
var new_project
var dest_img: Image
func menu_item_clicked():
        # Do some stuff
        if thing_to_do == 0:
                # get an image (For testing) from the current cel
                dest_img = extension_api.project.get_current_cel().get_image()
                # also make a new project
                new_project = extension_api.project.new_project([], "Test",
Vector2(64, 64))
        if thing_to_do == 1:
                # To change something or get something in a project we must make it
our "current_project" first
                extension_api.project.switch_to(new_project)
        if thing_to_do == 2:
                # Add 3 frames (the new_project will now have 4 total frames)
                for i in range(3):
                        extension_api.project.add_new_frame(0)
        if thing_to_do == 3:
                # Add a PixelLayer
                extension_api.project.add_new_layer(0, "I Love Pixelorama")
        if thing_to_do == 4:
                # Now change content at {frame 3, layer 1} of "new_project"
                extension_api.project.set_pixelcel_image(dest_img, 2, 0)  # {frame
3, layer 1}
        thing_to_do += 1
```

Custom Exporter

Adds a basic custom exporter that will export only odd frames.

```
extends Node

onready var extension_api: Node  # Variable for keeping reference to the Api

var id: int
```

```
func _enter_tree() -> void:
	extension_api = get_node_or_null("/root/ExtensionsApi")  # Accessing the
Api

	var exporter_info := {"extension": ".png", "description": "Only Alternate
Frames"}
	id = extension_api.exports.add_export_option(
		exporter_info, self, extension_api.exports.ExportTab.IMAGE, false
	)  # 2nd argument (in this case "self") must have "override_export()" in
it's script


func _exit_tree() -> void:
	# remember to remove things that you added using this extension
	extension_api.exports.remove_export_option(id)


func override_export(details: Dictionary) -> bool:
	# in this function you take control of image processing and saving from
pixelorama
	# return true for SUCCESS, false for FAILURE

	# the (details) include everything you could possibly need for exporting
	# keys of (details) are:
	# "processed_images", "durations", "export_dialog", "export_paths",
"project"
	for i in range(0, details["processed_images"].size(), 2):
		var image: Image = details["processed_images"][i]
		var error = image.save_png(details["export_paths"][i])
		if error != OK:
			return false
	return true
```

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Ways to contribute

Say something nice!

While material help is great, we don't want anyone to feel forced or obliged to give money to us, especially if they can't afford to. Sometimes saying something nice can go a long way! Showing your love and appreciation to someone could make their day, and give them strength to pursue their dreams and make their visions a reality. If you like Pixelorama and wish to help somehow, consider leaving a positive comment. And not just to the main development team, but to the awesome contributors who helped shape this software in so many ways! It's up to us to make a more wholesome world and every little bit counts. Of course, if you have more feedback and criticism to share, feel free to do so!

Spread the word!

Talk to your friends, co-workers, people from your communities and anyone who might be interested in Pixelorama! You can promote it on your social media, make videos about it or, even better, draw! Create content using Pixelorama and tell the world that you used it for your creations! Making Pixelorama more popular is a win-win scenario for everyone because more people knowing about it means that we will get more feedback, which will be used to make Pixelorama even better, and some people might be interested even in contributing code directly!

Donations

Pixelorama is a free and open-source project. This means that all of our income so far comes solely from donations. Unfortunately, while our creativity stems from our love of the project, today's world rules force us to make a living using real-life currencies. We need that money to survive and to keep our focus on our projects, such as Pixelorama, without having to worry about how to afford our rent and groceries by working at jobs that do not represent us. If you wish to see this project grow and help us financially, you can become our Patron and receive some nice benefits. You can also buy Pixelorama on itch.io, for an amount of your choice!

BECOME A PATRON

Contributing Code

You are free to open pull requests to Pixelorama's GitHub Repository to fix issues or implement new features. Keep in mind, however, that not all PRs will be merged. Some may need discussion, or others may be downright closed. Make sure to read the CONTRIBUTING.md file first.

Contributing Feedback

Do you have a super cool idea for the development of Pixelorama and want to share it? Then the best way to do it is by opening a Discussion on GitHub. That is the most effective way for your feedback to be seen and discussed with the team and other developers that are interested in helping.

Testing and reporting issues

Did Pixelorama crash? Did you find a bug, performance issue, visual glitches or overall unexpected behavior? If so, feel free to open an Issue on GitHub. If you wish to maximize the chances of the issue being addressed, please keep in mind to be as specific as possible and provide as many details as you can. The version of Pixelorama, your operating system, a clear and thorough explanation of the issue and steps to reproduce (if you know them) are a must to include. Otherwise, it will be harder for us to replicate the issue, and thus find the source of the problem. If you want to, including screenshots and videos of you replicating the issue in your issue report is a great help!

Contributing Documentation

The documentation is hosted on GitHub. If you wish to contribute, you are free to open pull requests. Make sure to read the CONTRIBUTING.md file first.

Contributing Translations

If you wish to contribute a translation, make sure to use Pixelorama's Crowdin page. Crowdin is a web-based translation platform, where people can contribute translations together and with ease. Please use Crowdin and do not directly open pull requests to Pixelorama's GitHub repository.

Link: https://crowdin.com/project/pixelorama

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

FAQ

If you have any questions about Pixelorama, there is a chance that someone else had the same question before, so it will be answered here, in the Frequently Asked Questions (FAQ). The question is indicated with "Q" and the answer with "A".

Q: How much does Pixelorama cost?

A: Pixelorama is available for the *extremely cheap price of $0.00!* That's right! Pixelorama is a free and open-source software (FOSS), available under the (very permissive) MIT license! In short:

- You are free to download and use Pixelorama for any purpose, personal, non-profit, commercial, or otherwise.
- You are free to modify, distribute, redistribute, and remix Pixelorama to your heart's content, for any reason, both non-commercially and commercially.

Q: What platforms is Pixelorama available on?

A: Pixelorama is currently available for Windows, Linux, Mac and Web browsers. More platforms may be supported in the future. You can download Pixelorama from itch.io and GitHub releases. It is also available on Scoop for Windows, as a Flatpak and Snap for Linux and in the Open Store for Ubuntu Touch.

Q: Who is working on Pixelorama?

A: Pixelorama is being developed by Orama Interactive, a game development team. Of course, Pixelorama would not be the same without the help of our beloved contributors!

Q: Is Pixelorama safe?

A: Pixelorama is completely safe, as long as you download it from official sources (see `Q: What platforms is Pixelorama available on?` above). Its open-source nature allows for perfect transparency, thus it is impossible for us to sneak in malicious code without anyone noticing. The official builds are also being generated directly from the source code using

GitHub Actions, so there is no worry that someone is sneaking in malicious code in the process of compiling. You may, however, get some security warnings from security software, such as Windows SmartScreen. The reason for this is that Pixelorama is not signed, and thus not recognized by the security system. You can safely ignore these warnings and proceed to run Pixelorama.

Q: What engine is being used to develop Pixelorama? Where is the code hosted?

A: Pixelorama is being developed with the Godot Engine (version 4.1), and since it is open-source anyone can contribute to it! The code is available on Pixelorama's GitHub repository!

Q: Can I contribute to this project?

Yes! Read our Ways to Contribute page to learn more.

Q: Will Pixelorama ever become Godot's Plugin?

A: Pixelorama was originally created as a stand-alone program. Our current focus is to keep improving it and not implementing it into Godot itself, but we are not declining the possibility of it being a plugin one day. However, if you are interested in direct Pixelorama support from Godot, you can use this amazing Godot Pixelorama importer plugin which can be used to make Godot import pxo files.

Q: What languages is Pixelorama available in?

A: The main language is English, but Pixelorama is also available in numerous other languages! All of our translation files are hosted in Crowdin. By default, the language that Pixelorama is in is the system language of your device. You can easily change it by going to the Edit menu, Preferences, Language.

Q: What are the minimum system requirements to run Pixelorama?

A: Pixelorama is a relatively lightweight software and it should run on most devices, as long as OpenGL 2.1 / OpenGL ES 2.0 is supported. Please be aware that once Pixelorama v1.0 is out, these requirements will be increased to OpenGL 3.3 / OpenGL ES 3.0.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Compiling from Source

To compile Pixelorama's source code, all you have to do is download Godot 4.1. You don't need the mono version since Pixelorama is written entirely with GDScript. Then, you clone Pixelorama from its GitHub repository, load the project within Godot, and run it! Simple as that!

If you wish to export Pixelorama as a standalone executable to a platform, you can refer to the relevant page of the documentation of Godot:
https://docs.godotengine.org/en/stable/tutorials/export/exporting_projects.html

*Last updated on **Apr 27, 2024** by **dependabot[bot]***

Troubleshooting

This page lists common issues encountered when using Pixelorama and possible solutions. To experience as less issues as possible, make sure you are always using the latest stable version of Pixelorama.

Pixelorama crashed while I was working on my project!

Crashes are unfortunate, and while we are doing our best to fix them, some issues causing the application to crash may have escaped our grasp. The good news is that you can enable **automatic backups**, that save your work every a certain time interval. You can enable them from the Edit menu, Preferences, Backup and then by ticking on "Enable autosave".

Besides that, it is **strongly recommended** that you save often, and keep copies of your files as backups in case something gets lost.

You can also aid us in our fight against crashes by reporting issues on GitHub and uploading the content of Pixelorama's log files. You can find the log files in `C:\Users\[USERNAME]\AppData\Roaming\Pixelorama\logs` on Windows, `~/.local/share/Pixelorama/logs`

on Linux and `/Users/[USERNAME]/Library/Application Support/Pixelorama/logs` for macOS. It is also extremely helpful, if not necessary, to be as specific as possible with the steps that led to the crash. By giving us specific steps to reproduce the issue, we can solve it faster.

My images are being saved as pxo files, and not png, jpg, gif, etc

You are saving instead of exporting. Saving creates pxo files, which are Pixelorama's custom file format for projects. To create image files such as png, you need to go to the File menu and select Export. You can refer to the Save and Export page for more information.

macOS says that Pixelorama is damaged!

You may encounter some errors saying that Pixelorama is damaged. This is because the latest versions of MacOS mark the launcher of unknown binaries as a non-executable file. This most likely has to do with the fact that Pixelorama is not signed.

Read this guide for more information: https://disable-gatekeeper.github.io/

I have performance issues, Pixelorama is very slow on my machine

We are doing our best to optimize the performance. That being said, you can improve the performance of the app significantly by changing its maximum framerate, in order to reduce CPU usage. To do that, you can go to the Edit menu, Preferences and then click the Performance tab. There, you can set a maximum value for the FPS in "Set application FPS limit:". Keep in mind that the lower the number, the lower the CPU usage, but the application gets slower, choppier and unresponsive, so make sure not to set it *too low*. 0 means that there is no limit.

If you have a specific performance issue you'd like us to look at, feel free to open an issue on Pixelorama's GitHub Repository.

I have a hiDPI monitor and the User Interface is very small!

It is possible to change the scale of the UI, by going to the Edit menu, selecting Preferences and going to the Interface tab. There, you can change the display scale, which goes from 1 (smallest) to 4 (biggest). 4 essentially means that the UI will be 4 times bigger than the default UI. Then, you click Apply to set the changes. Keep in mind that it's best to use integer values, such as 1, 2, 3 and 4, otherwise the UI may appear blurry.

Error code 12 when exporting png files

This error has only been spotted in the Windows platform. Sometimes, when exporting png files, the process may fail due to "Error code 12". This most likely happens when you try to export in a directory that is protected by **anti-virus software**. To resolve it, you can try tweaking the settings of your anti-virus to either whitelist Pixelorama or disable folder protection. We can't guarantee that this will work, as it also depends on what anti-virus software you are using. Alternatively, you can export in a directory that is not protected by the anti-virus.

It is also possible that this issue may be solved if you run Pixelorama as an administrator.

If neither of these works, you could save your projects as a pxo file, open the Web version, drag and drop the pxo file there, and export the png file from the Web version.

Drawing is very slow when using a drawing tablet

If you are on Windows, try changing the tablet driver Pixelorama is using, from the Edit menu, Preferences, Drivers.

Guides are not being created

To create guides, you have to drag them out from the rulers, on the top and the left of the canvas. However, sometimes this doesn't appear to do anything. If that's the case, then it is very likely that you have disabled guide visibility. To make sure that they are visible, go to the View menu and see if "Show Guides" is ticked on. If it is not, make sure to tick it.

*Last updated on **Apr 27, 2024** by **dependabot[bot]***