Shaoqian's Proposal: K12 Mode for Purr Data

Zhang Shaoqian

April 9, 2019

Contents

1 Meta Info

Name Zhang Shaoqian

Email [redacted]

Education Year 2 Computer Science, National University of Singapore

Purr Data GitLab @nerrons

GitHub @nerrons

Timezone UTC+8

2 Project Overview

The task of porting K12 mode to Purr Data is described in the Summer of Code ideas list page. The focus is mainly on the frontend: adding a friendly user interface that allows effortless interactions with existing K12 abstractions. Besides, to further improve usability, full touchscreen support will be added as well. The proposal can realized in an incremental manner, frequent pull requests safely merged into the master branch throughout the summer.

2.1 Deliverables

- An easy way to start Purr Data in K12 mode on all platforms
- K12 abstractions that can be used out of the box
- A K12 frame containing all abstractions next to the canvas
- Menubar, pop-ups and keyboard shortcuts designed for K12 mode
- Tooltips for xlets of K12 abstractions
- Touchscreen support for K12 mode and potentially the normal mode

3 Implementation Details

This project can be done incrementally. When there are multiple solutions to one problem, I will realize the most simple yet effective one first and then improve them depending on the results of the first iteration.

3.1 Make an entry for K12 Mode

First, write a function in `pdgui.js` that allows setting `k12_mode`, and in `set_vars()` of `index.js` call that function with `argv`. Then, find ways to add shortcuts or start menu entries for all platforms.

We also need to copy the whole `l2ork_addons/K12` folder to the installation folder in a certain `makefile`, which will not be difficult to do.

3.2 Fix K12 abstractions

K12 abstractions are abstractions made specifically for K12 mode, typically squeezed into a GOP area with back ground images. The good news is that all K12 abstractions can work with the latest Purr Data without modifying any code; thus only appearance issues remain.

Adjust the GOP area

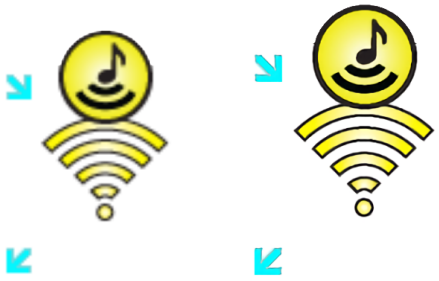
Currently, if we load up a K12 abstraction with default zoom level on Windows, the bottom and the right borders are cropped. When we zoom in or out, however, those borders will show up again. This might indicate a bug in GOP rendering and needs to be looked into. If no solution can be found, we can just make the GOP area in each abstraction slightly bigger.

Enhance background images

The original resources for background images are in very low resolution and look bad when the canvas is heavily zoomed-in. There are two ways to address this.

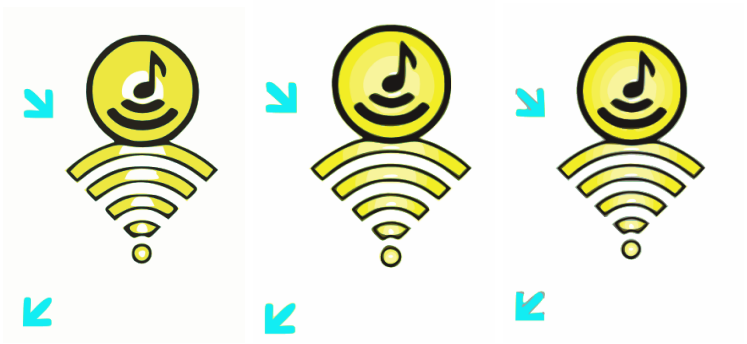
- Upscale the original PNGs to higher resolution, as shown in Figure 1, then use `[draw image]` to fit them in the GOP area. Upscaling such simple pictures produce nice results so this method is definitely feasible.
- Jonathan suggested we convert PNGs to SVGs and use them instead. The resulting SVGs also have good quality and small size, as shown in Figure 2. However, we must find ways to display SVGs with very small selection areas, since `[image]` does not directly support SVGs.

Overall, both methods will greatly improve the quality of background images. I will implement the former one first and then upgrade to the latter if needed.



(a) Original PNG file. (b) Upscaled 4x.

Figure 1: The original and upscaled version of K12 abstraction background images.



(a) 4 scans. (b) 7 scans. (c) 10 scans.

Figure 2: Different versions of the SVGs converted from the PNGs.

Add help patches

The help patches should get indexed in `make_index()`. Since all patches come with well-written `[pd META]`, it won't be hard. Right-click popups will work without changing anything.

3.3 Set up `[preset_hub]`

A `[preset_hub]` is needed in every K12 patch for the Preset abstraction to work. We already have the code that adds one `[preset_hub]` in all new patches; we just need to skip drawing them in `gui_gobj_new()` whenever the object is a `[preset_hub]` and K12 mode is on.

3.4 Add K12 frame

The panel

The panel is an area that contains all the buttons. It will be a `<div>` sitting right next to the main `<svg>` on canvas windows, styled with "`height: 100%; position: fixed;`". If NW.js gets upgraded, it can also be implemented with CSS Grid.

Because of the width of the panel, there will be a horizontal bias when clicking and drawing, and thus offsets for those events are needed in the JS code. A working prototype demonstrates how this problem can be addressed; please refer to Section 4.2.

Other geometry issues are not as urgent, including setting a minimum height and width, making sure zooming works, etc.

The buttons

The layout of the buttons will be defined using HTML tables or CSS Grid, either hard coded into HTML or added by JS. It consists of:

- A big button that toggles Editmode. The pic on that button should change when clicked to reflect the current mode.
- Two tabs for choosing categories: Control or Sound
- Many buttons of K12 abstractions from the current category, listed in a predefined order.

Tooltips will be available for each button using the `title` attribute. The result should be close to the original K12 mode, as shown in Figure 3.

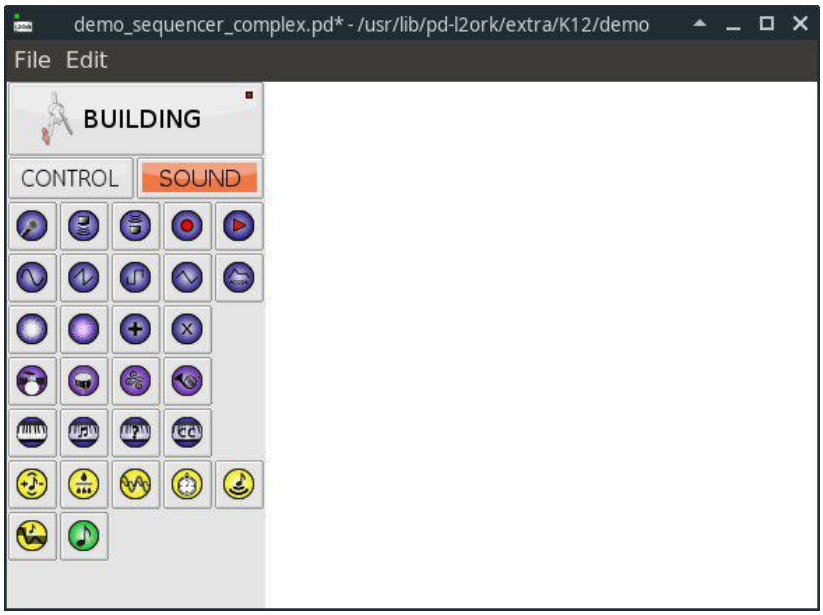


Figure 3: The layout of buttons in the original K12 mode.

Toggle on/off

When a button (outside of the panel) is clicked, the whole frame will hide/appear. There are two possible designs:

- The frame sits next to the canvas. When toggled on, it pushes the canvas to the right a little bit.
- The frame floats above the canvas. When toggled on, it gently covers the canvas on the left side.

Both should be easy using a "`display: none;`" style, and even better, with CSS animations. The horizontal bias will need to be taken care of each time the user toggles the frame.

Other UI stuff

- Keyboard shortcuts: To be added to `pd_shortcuts.js`.
- Localization: This requires the buttons to be defined using JS. There are plenty of examples how to do this in the existing code base.

3.5 Change the menubar structure

These changes will happen in `pd_menus.js` and the menu part of `pd_canvas.js`:

- In File, add "Show K12 demos". This can reuse the code for `m.file.open` in `nw_create_pd_window_menus()`; just tweak the directory in `build_file_dialog_string()`.
- Wrap up any unwanted entries with `if` statements. For example, anything related to "find".

Of course, there can be further discussion about what menus need to be removed, or whether a menubar should exist at all. Most changes will just be within the JS files mentioned above.

3.6 Support tooltips in canvas

Tooltips for xlets

When the cursor hovers over an xlet, a tooltip explaining what this xlet does must be shown. Since xlets are `<rect>`s, a child `<title>Tooltip Goes Here</title>` will be the most suitable way. Using CSS tooltips will make it very difficult to ensure the tooltip shows up within the window, and using JS is just not worth it. Then it seems simple: when creating the `<rect>`, shove in a `<title>` child. Unfortunately, currently when creating that `<rect>`, the frontend does not know which kind of K12 abstraction it belongs to, let alone the content of the tooltip.

The JS code in charge of drawing xlet `<rect>`s is a `create_item()` inside `gui_gobj_draw_io()`, which needs to know which type of K12 abstraction is being drawn. That knowledge will come from either of these two sources:

- The front end. Specifically, `gui_gobj_draw_io()` can fetch the info from the parent `<g>`, which needs another attribute to remember the type of the abstraction it represents, and then map the type of the abstraction into the content of the tooltip. Then we must modify some C code, including:
  - `canvas_obj_abstraction_from_menu` in `g_text.c`, since we `pdsend` about "`obj_abstraction`" when creating K12 abstractions on canvas;
  - Relevant code `gui_vmess()`ing `gui_gobj_new`.
- An additional field in the `gui_vmess()` calling the `gui_gobj_draw_io()`. This may require more changes in C code because it sounds like we need to pass down the information through a series function calls.

Either way, we need to pass the type of the K12 abstraction through the core. I prefer the first method because it's less amount of messages to be sent back and forth.

Tooltips for abstractions

Tooltips are also shown when the cursor hovers over the K12 abstraction itself. This is in fact more tricky to do than xlet tooltips. Since the background image is the top layer excluding other iemguis, only `<title>`s inside the `<g>` that includes the `<image>` will work. However, the images are much larger than the GOP area, so the tooltip will show up even when the cursor is not inside the body of the abstraction, as shown in 4. Then we have two options:

- Make the background image as small as possible.
- Use JS to determine whether the cursor falls inside the GOP area.

Both options do not seem elegant. Besides, one can argue that tooltips for abstractions are not necessary at all, since the K12 frame on the left will provide the same information, and showing a tooltip whenever the mouse wanders around can be sometimes annoying (for me that is a source of distraction).

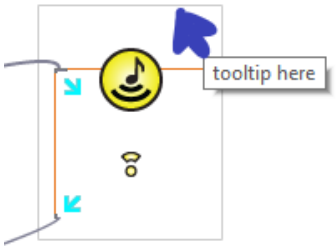


Figure 4: The wild tooltip for K12 abstractions. No idea how to capture the cursor.

Source of the tooltip content

As Jonathan has pointed out, the old K12 mode parses the help patches for the content of the tooltip. Since we aim to consolidate the tooltip when we draw an abstraction, we need to build a dictionary-like structure at startup. Two ways to do that:

- Parse all `-help.pd` files, store all meta information in a JSON object, and directly retrieve them later on. The parsing process will not be hard to come up with because we already have Tcl code for that. This approach is viable because there aren't many K12 abstractions and all descriptions are short.
- Utilize Elasticlunr.js, index `-help.pd` files at startup (which we probably will do anyways), and search them when an abstraction gets created.

Honestly, I don't see any significant advantage of the index method over the parse-it-all method, so I will definitely try out the parse-it-all method first.

3.7 Support touchscreen

Jonathan also suggested supporting touchscreen. I have a working Windows laptop with touchscreen, so substantial testing can be achieved.

Translate to a mouse

Translating touch events to mouse events should be the easiest way to get touchscreen working. Just use `MouseEvent()` and generate mouse events on the same spot accordingly, as following:

- `touchstart` → `mousedown`
- `touchmove` → `mousemove`
- `touchend` → `mouseup`

Enhance precision

However, the method above will not work smoothly. Because the xlets are very small, if we directly translate touch events to mouse events, the simulated click might never land on the xlet the user wanted to touch. There are, apparently, two ways to deal with that:

- Check if there are xlets around the touch point, and "snap" the coordinates of the generated mouse event to those. If there are multiple xlets, choose the closest one. Also, we may define rules to reduce computation, e.g. a touch can only snap to an outlet when it comes from an inlet, etc. This probably won't require any changes in C code, but performance can be a problem.
- Make xlets bigger. When the app detects a `touchstart` event, make the selectable parts of xlets bigger without changing the appearences of `<rect>`s. This will probably have something to do with the Pd code, since currently the core checks what exactly a click lands on, but should be faster.

The incremental style can be applied here, again.

3.8 Wii & Arduino support

I do not own Wii or Arduino devices, therefore cannot test whether the Wii and Arduino abstractions are working. Nevertheless, I am pretty confident they will, as the relevant code in C did not change. If I can get my hands on these devices in the future, I will definitely ensure they work.

4 A primitive prototype that works

As shown in Figure 5.

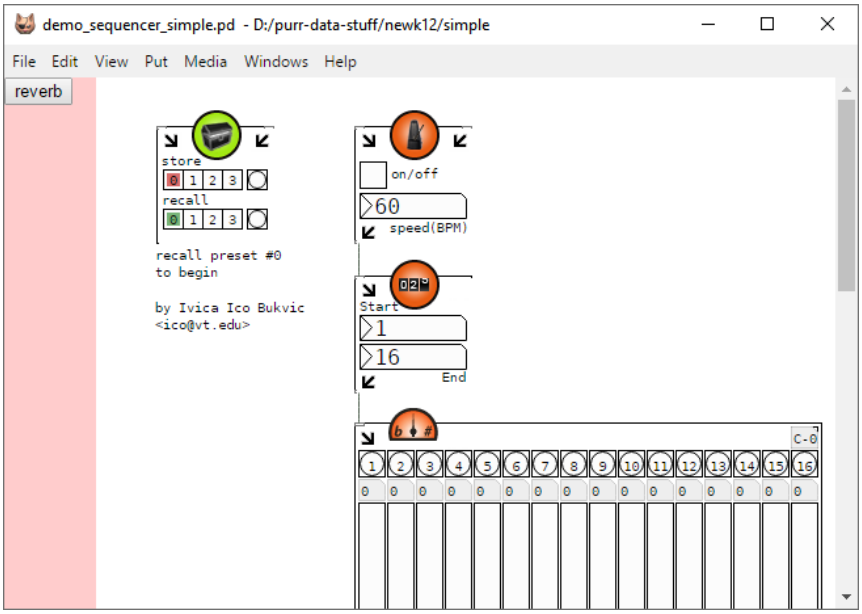


Figure 5: A screenshot of the K12 prototype.

4.1 To try it out

1. Install the latest release of Purr Data.
2. Copy the whole `l2ork_addons/K12` folder from the Purr Data repo under `pd-installation-folder/extra`. The installation folder is usually:
   - On Windows: `C:\Program Files (x86)\Purr Data`
   - On macOS: `/Applications/Pd-l2ork.app/Contents/Resources/app.nw`
   - On Linux: `/usr/lib/pd-l2ork`
3. Back up these files from `pd-installation-folder/bin` and replace them with the files here:
   - `pd_canvas.html`
   - `pd_canvas.js`
   - `pdgui.js`
4. Start Purr Data as usual and create a new patch to see the effect.

4.2 Explanations

The exact changes can be found in this diff.

`pd_canvas.html`

At the same level of the main `<svg>`, a `<div>` is added, containing a button that shows a tooltip when hovered (Figure 6) and sends to the core a message about creating a new K12 object when clicked. At the top, a `<style>` is added to place the `<div>` on the left and move the `<svg>` to the right.

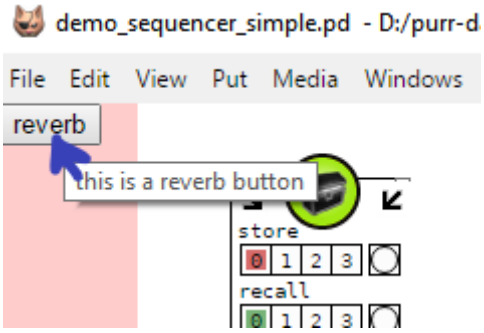


Figure 6: Tooltips of the K12 frame. No idea how to capture the cursor.

`pd_canvas.js`

After the K12 frame is added to the window, the X coordinate from the `pageX` attribute and that from the Pd's own coordinate system become different, the former greater than the latter by the width of the frame. Thus, offsets are added to mouse events to unify them so the intended position is "clicked".

`pdgui.js`

Offsets are added to functions that simply draw something on the canvas based on given X and Y coordinates. Things drawn are moved to the right by the width of the K12 frame so they follow the mouse cursor. Also the magic number controlling the minimum width of the canvas is changed to another magic number, so the horizontal scrollbar do not appear by default.

4.3 Some Known Bugs

- When opening existing patches whose canvases have saved widths and heights, the canvas is pushed to the right by the K12 frame.
- When placing a K12 object from the K12 frame, the abstraction can be placed to the left of the canvas, under the K12 frame.
- Pressing ESC when placing K12 objects has the same behavior as left-clicking with the mouse, which is different from normal objects' behavior (does nothing).
- ...

5 Proposed Timeline

The whole project will be divided into 12 weeks. The last week of each coding phases (Week 4, Week 8 and Week 12) are reserved as a "buffer week" dedicated to finishing up jobs, testing, fixing bugs and writing documentation. At the end of buffer weeks, the code is expected to be merge-ready.

For each week, I aim to finish the proposed goals before weekend and then invite my mentor to review the changes. If the project progresses faster than planned, I will spend time on other aspects of Purr Data as well, like the font issue.

The Prologue

- Continue exploring the prototype, try out possibilities mentioned in the implementation details and see what's best.
- If we decided to upgrade NW.js, help the community get it done.
- Fix more bugs, especially GUI related ones. For example, #447.
- Of course, actually start doing stuff if I have time.

Week 1 (Phase One starts)

Get all "meta" stuff done. Goals:

- Changes made by the prototype can be toggled using the `-k12` flag.
- `-k12` works together with other flags.
- All platforms have convenient ways (shortcuts, etc.) to start Purr Data in K12 mode.
- Nothing in the build system needs to be changed since then.

Week 2

Solidify the layout of the K12 frame with HTML. Goals:

- The layout is close to the original K12 mode.
- All buttons on the frame work.
- The code has potential to support localization.

Week 3

Make K12 mode work. Goals:

- The `[preset_hub]` is hidden.
- The menubar has only necessary entries.
- If there is time, work on UI improvement goals for Phase Two.

Week 4

Buffer week.
For the 1st Evaluation: A working K12 mode with the complete frame. It might not be the best looking one, but it will allow building arbitrary K12 patches.

Week 5 (Phase Two starts)

Work on xlet and abstraction tooltips. Goals:

- Tooltips work. There is a minimal amount of changes in the C code, hopefully only in the layer between the core and the GUI.
- Content of tooltips are stored in an efficient way.

Week 6

Enhance image quality. Goals:

- Taking the upscaled PNG path:
  - All images are upscaled.
  - All abstractions have new PNGs fitted into the GOP area.
- Taking the SVG path:
  - Either `[ggee/image]` or `[draw svg]` work.
  - All images are converted to SVGs.

Week 7

Finish up UI improvements. Goals:

- K12 mode has a new set of keyboard shortcuts, similar to but simpler than the original one.
- The K12 frame can be toggled on/off.

Week 8

Buffer week.
For the 2nd Evaluation: All features in the original K12 mode are ported to Purr Data. At this stage, K12 mode should look attractive.

Week 9 & 10 (Phase Three starts)

Implement touchscreen support. Since this would require a lot of testing, it is safer to assume two weeks are needed. The only goal:

- Touchscreen can do anything that a mouse can do—easily.

Week 11

Write a K12 compressor and a new, minimalist demo patch using that compressor. Update the frame accordingly.

Week 12

Buffer week. If time allows, continue working on the compressor. Then close #16 :-)
For the Final Evaluation: Full touchscreen support and new abstractions/patches.

6 About Me

6.1 ...and Purr Data

Under the name `@nerrons` and the email address `coniinedieu@gmail.com`, I have a few elementary modifications merged into master (!257, !260 and !270) since January, the last one about our building system (which "humans cannot reason about", according to Jonathan). After the initial merge requests I took my time reading and experimenting with the code base, including everything related to NW.js, the legacy TCL code, the C code on communication between the core and the GUI and a bit of the implementations of the core. Then I decided to pursue porting K12 mode as I have more experience with the front end. Here is the mailing list thread where I consult Jonathan about the rough plan I posted on March 26.

6.2 ...and Open Source Software

I am not an expert, but I have dabbled in and am passionate about Open Source.

- Warship is a crude audio engine written in C++ that supports 3D audio and sound virtualization. It uses low level APIs from FMOD for playback and wraps channels in state machines, allowing more control on individual sounds.
- React Business Tool is a demo I created when learning React. Though simple, it is thoroughly tested and strictly follows the style guide. It looks like this on Heroku.
- A hotel management system, a medium-scale Java app, is made for a course at my school (Software Engineering) in a 4-pax team.

Also, I value communication: I always try to make myself understood, whether it is about opening issues, writing to the mailing list or writing comments and documentation. I think this proposal and my posts in the mailing list already provide good examples.

6.3 ...and Music

One of the biggest reasons why I'm excited about Purr Data is that I make music myself. I have several albums (which no one listens to) released on Bandcamp, Spotify and Apple Music. Always toying around with synthesizers, I'm also a regular user of Max which, well, makes it easier to fall in love with Pd. Besides coding, I'm very confident that I can also make quality library patches for Pd as I gain more experience.

6.4 ...and GSoC

- This is the only proposal I've submitted.
- Luckily all my exams end before [redacted], so throughout the summer I have at least 35-40 hours per week.
- I can foresee myself contributing to Purr Data even after the summer ends, hopefully becoming a long-term contributor/maintainer. Pd is just beautiful and I don't see any reasons to keep my hands off it. Hopefully I can kick-start with GSoC, and thank you very much for taking time to read this proposal.