CS4624: Multimedia, Hypertext, and Information Access

Final Report

Pd-L2Ork

Professor: Edward A. Fox
Client: Ico Bukvic
Team Members: Drew Bowman, Lily Khochareun, Ethan Springs, Sahar Farzanehpour

VIRGINIA TECH, BLACKSBURG, VA 24061, MAY 8, 2023

Contents

List of Figures

List of Tables

1 Abstract

Over the past several years Professor Ico Bukvic has been developing his own extension to the visual programming language of Pd, called Pd-L2Ork. This software was designed to take the functionality of Pd, a computer music programming software system, and apply it to Virginia Tech's Linux Laptop Orchestra, or L2Ork. Since 2010, it has been used extensively in education, research, and production. Another product of this research is the K12 mode that Pd-L2Ork has more recently introduced with the help of a prior CS Capstone Team [1]. K12 mode is targeted specifically at beginners, to introduce sound programming and design, as well as for elementary to high-school students.

Our team's job was to expand on this existing software in two distinct ways. The first way we expanded the software's functionality is through the implementation of object tooltips. Since Pd-L2Ork is a visual programming environment, the objects you create in it take up physical space and can be hovered over. Each object (or patch) has a series of inlets and outlets that can be used to pass data into an object and receive output from an object, respectively. Each of these inlets and outlets require specific types of data to be passed into them for different uses. You might call the values passed into inlets parameters to a function/method. Consequently, you might call the values received through the outlets of an object the return value of a function/method. The way that the programmer identifies what needs to be passed into each inlet and what is received through each outlet is through tooltips. Hover over one of these inlets/outlets to view the object's tooltips, which describes the type of data the inlet needs and what the data should describe.

Our team's task was to take the descriptions of each object's parameter(s) and return value(s) from the object's documentation and create tooltips for each object in every patch. This task was composed of two parts: tooltips for an object itself and tooltips for inlets and outlets of every object. In order to accomplish the first part, we take the description of the object out of the index and attach it as a title tag to the particular object. For the second part, we needed to parse in the information of the inlets and the outlets of the objects from the index and also attach it as a title tag to the particular inlet or outlet. The deliverable for this task was a merge-able code branch with code to enable this feature, which we gave our client access to.

The other task of our team was to integrate a button that moves a Pd-L2Ork patch onto a web browser. This first involves compiling the original Pd-L2Ork code (written in C) to Web

Assembly through a tool called Emscripten[2]. This took up the majority of our time because of the large code base and many errors found by the Emscripten compiler that needed to be corrected. Once we edited the code base to make it Emscripten compatible and compiled it, we included the generated JavaScript file in our front-end code and linked the audio back-end to the front-end on the browser. This was a difficult task, but our team gained valuable insight from similar examples in PdWebParty[3] and Purr-Data[4]. In order to accomplish this task, we performed the necessary research to begin implementation, solved problems through file comparison and classic debugging strategies, and we asked Dr. Bukvic for help promptly when we needed it. The deliverable for this task was a button that could gather up all necessary files to display a patch current in Pd-L2Ork onto a web browser. We did not complete the work needed for this deliverable, but have provided documentation (in the sections for the Developer's Manual and on Future Work) regarding what has been done by the team, and what can be done by future teams to reach this deliverable.

2 Introduction

Pd-L2Ork is an open-source software system designed for real-time interactive multimedia performances. This environment is largely based on the Pure Data (Pd) language and developed by our client, Dr. Ivica Bukvic. This powerful tool is used by a diverse range of people from professional musicians to educators and students. Virginia Tech's Linux Laptop Orchestra, L2Ork, is one of the many organizations and individuals that make use of the real-time audio and visual processing features to create unique and innovative performances for their audiences. [5] Pd-L2Ork has a GUI that provides non-programmers with a usable environment for creating different sounds and manipulations. In the Fall 2022 semester, a K12 mode was implemented to help new users better understand how the tool is used. [1] Usability is a primary goal for the developers of Pd-L2Ork, so for this project we aimed to continue working toward this goal by implementing object tooltips and a web browser button.

2.1 Objectives

The primary objective of this project is to improve overall usability of this tool and build upon the pre-existing code to further Pd-L2Ork's goal of providing a usable environment for creating different sounds.

2.2 Deliverables

Our first deliverable is implementing tooltips so a user can hover over an object and obtain information on that object. We also plan to implement a web browser button which will allow users to export an existing patch that they've created in Pd-L2Ork to use in a web browser.

2.3 Client

Our client is Professor Ivica Ico Bukvic, who is the founder and director of the Digital Interactive Sound and Intermedia Studio (DISIS) and Linux Laptop Orchestra (L2Ork). His principal areas of research include Immersive Audio, Sonification, New Interfaces for Musical Expression, Recontextualizing STEM K-12 Education, and Arts + Health. [6]

2.4 Team Members

Our team consists of Drew Bowman, Lily Khochareun, Ethan Springs, and Sahar Farzanehpour.

3 Requirements

Our client Dr. Bukvic had two main tasks for our team to complete:

1. Tooltips
2. Web Browser Button

Both tasks focus on improving the usability of Pd-L2Ork as a whole. The end goal is to create a patch containing both tasks that can be cleanly merged with the main Git [7] repository.

3.1 Tooltips

The objective of this task is to develop a feature that provides information on a particular object by hovering the mouse over it. All the relevant data is already present in the database, and our task is to devise an algorithm that facilitates the transfer of this data. To accomplish this, an engine must be created that extracts metadata and appends it to the tooltip. Although parsing the information is mostly complete, an index argument must be incorporated.

3.2 Web Browser Button

This feature will give users the ability to export patches created within Pd-L2Ork to an external internet browser. The goal is for this functionality to be seamless for the user, only requiring the click of a single button. To accomplish this, we must carefully merge and edit existing code from various branches of Pd-L2Ork into the main Pd-L2Ork source code. Initially, we will attempt to execute this functionality in Emscripten, which is a comprehensive toolchain for WebAssembly [2]. Subsequently, we will run it in PdWebParty, and ultimately, integrate the implementation into the Pd-L2Ork code base.

4 Design

After our team understood the requirements at hand, we outlined our design plan for the tooltips and web browser button. We created two graphics, shown in Figure 1 and Figure 2, that break down the two major tasks into smaller sub tasks.

4.1 Tooltips Design

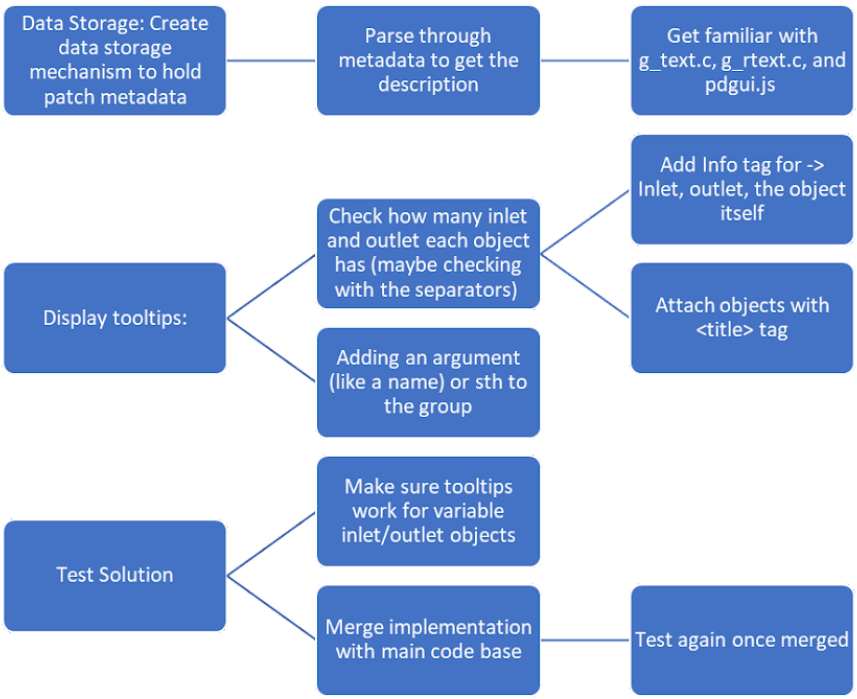

Figure 1: Design for tooltips functionality

4.2 Web Browser Button Design

Figure 2: Design for web browser functionality

5 Implementation

5.1 Tooltips

This task was composed of two different parts: the tooltips for the whole object and the tooltips for inlets and outlets of every object (see Figure 3) This would allow users to hover over an object whence a description of the object is displayed, and also hover over an inlet or outlet whence a description of the specific inlet/outlet is displayed.

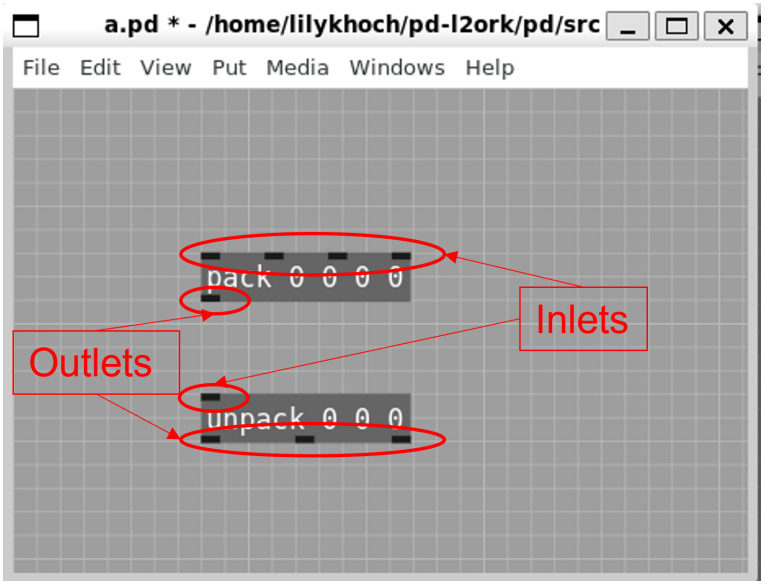

Figure 3: The difference between inlets and outlets.

The first step to start the implementation was to figure out what files we needed to edit. Our main focus was on a JavaScript file (pdgui.js) which takes care of displaying and also a C file (g_text.c) which takes care of parsing documentation and storing the data. We started by studying these files to better understand the creation of objects as a whole. After this, we hard-coded some tooltips, shown in Figure 4, in the HTML using the developer tools of Pd-L2Ork to better visualize the final product of the tooltips implementation.

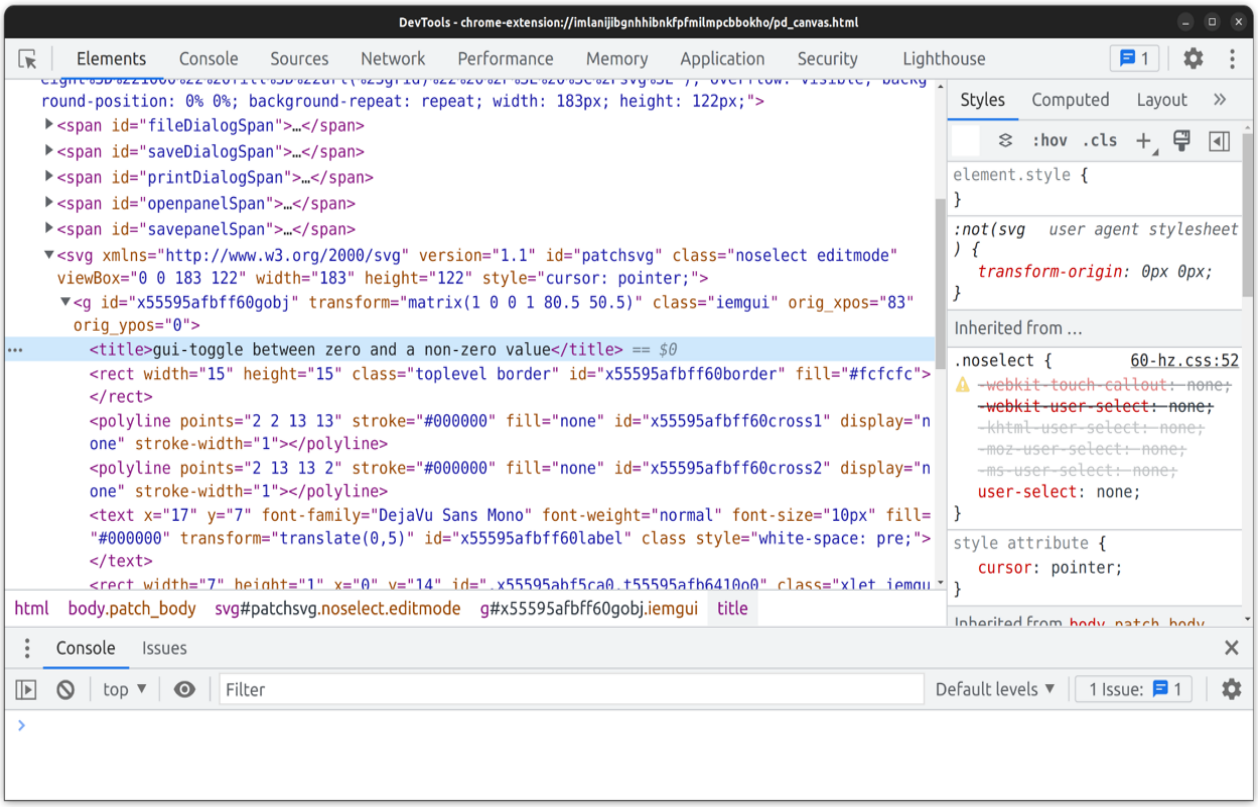

Figure 4: A manually added title tag.

For the first part, tooltips for the whole object, we edited the gui_gobj_new function by adding a parameter to pass in the name of the object. From this, we were able to edit some files on the back-end (g_text.c) to pass the name of the object to the front-end. This allowed us to search the index with the name of the object to find the description of the object. An index contains multiple fields that help identify the object (see Figure 5). For the purposes of this part of the task, we only use the title and description fields.

```
function init_elasticlunr()
{
    index = elasticlunr();
    index.addField("title");
    index.addField("keywords");
    index.addField("description");
    index.addField("related_objects");
    index.addField("ref_related_objects");
    index.addField("inlets_outlets");
    index.setRef("id");
    return index;
}
```

Figure 5: Index fields.

The parsing of the index was already done for us, so we were able to easily search for the description to set the contents of the title tag and attach the title tag to the object. In addition to searching the index, we also added another attribute to the object called "obj_text" which holds the name of the object. This is used later for the inlet/outlet tooltips.

```
if (tipFirstArg.length > 0) {
    var x = document.createElementNS("http://www.w3.org/2000/svg", "title");
    var t= index.search(tipFirstArg,{fields: {title: {}}});
    if (t.length > 0) {
        var yyy = index.documentStore.getDoc(t[0].ref);
        x.textContent=yyy.description;
        g.appendChild(x);
    }
}
```

Figure 6: Code that adds the title attribute to the object.

Figure 6 shows this process. We first create a title element and search for the "tipFirstArg" (name passed in from the back-end) in the index. The variable "t" contains an array of all the objects that contain the "tipFirstArg" as the title. We only use the first element of this array. Variable yyy is set to the index of the specific object. We set the title content to the description from the index. This process added a title tag to all objects at creation time. As is shown in Figure 7, when a user hovers over the object, the description is shown.

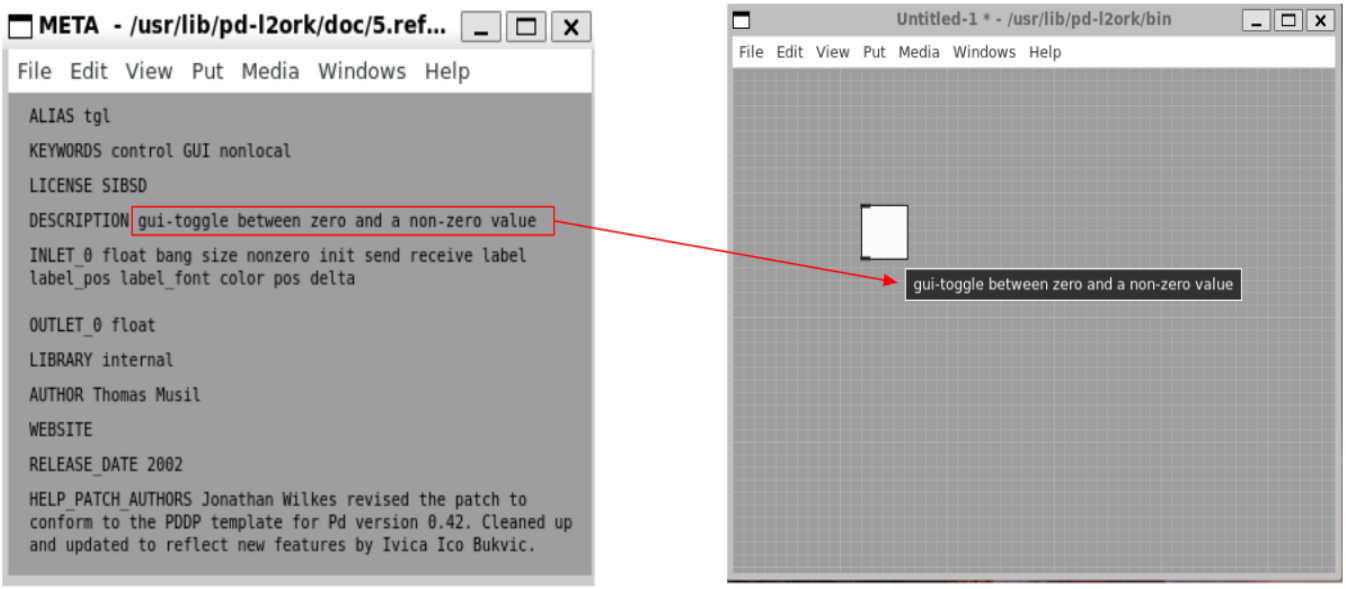

Figure 7: The correlation between the META file and the hoverable tooltip.

The second part of this task was more challenging since the parsing was not done for us. For this we had to edit the index to handle another attribute for inlets and outlets. In Figure 5 we added another field to the index called index_outlets. To set this attribute, we had to parse the pd help files for all the inlets and outlets using a regular expression (regex) shown in Figure 8.

```
var inlet = big_line
    .match(/#X text \-?[0-9]+ \-?[0-9]+ INLET_. ([\s\S]*?);/gi);

var outlet = big_line
    .match(/#X text \-?[0-9]+ \-?[0-9]+ OUTLET_. ([\s\S]*?);/gi);
```

Figure 8: Regular expression to read inlet/outlet lines from the pd help file.

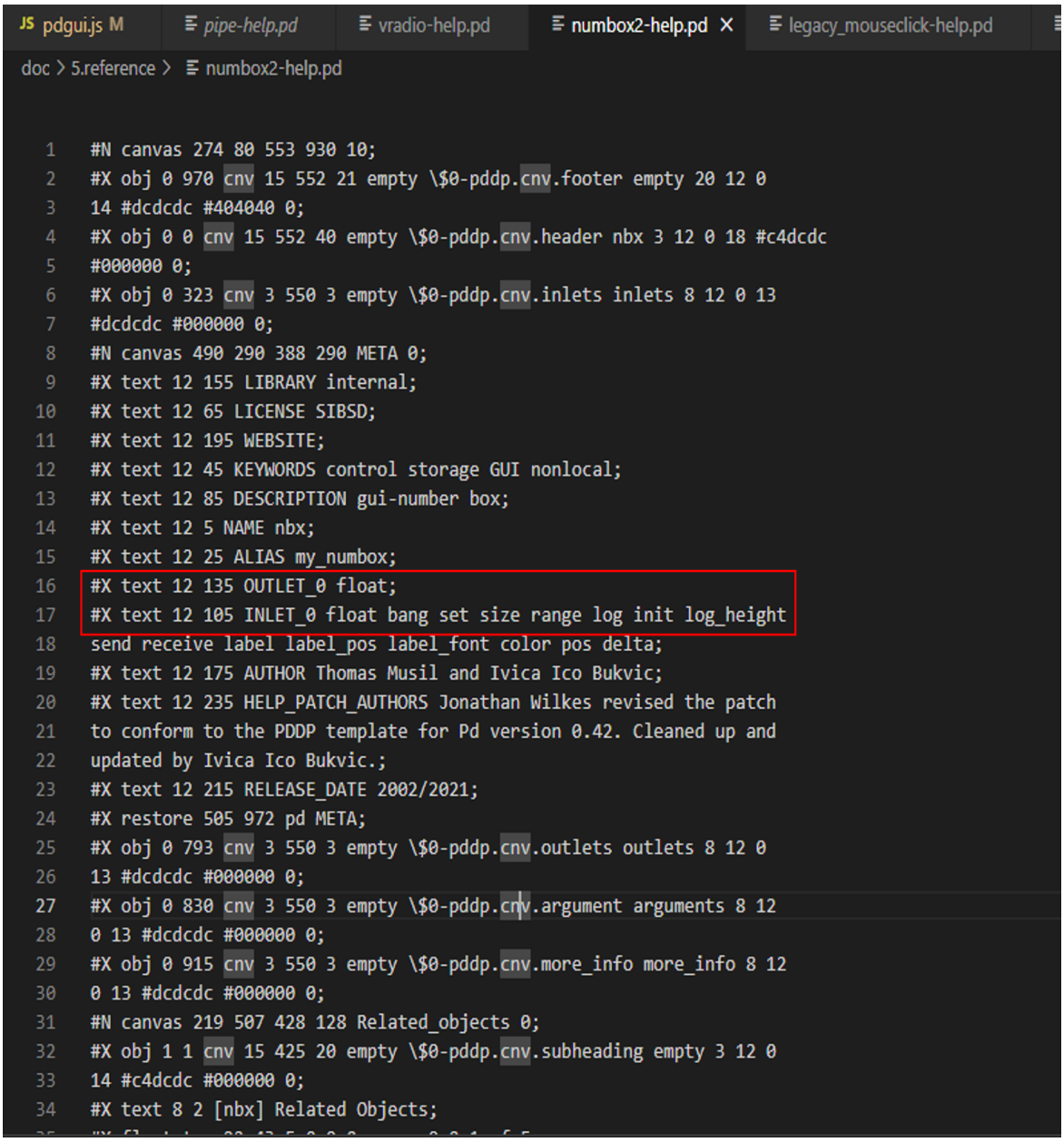

Figure 9: pd help file for the object numbox2.

In Figure 9, boxed in red is the line that the regex matches with. After parsing, we stored them in a pipe separated string with each inlet/outlet pre-pended by an IX/OX where the first character indicates whether it is an inlet or outlet and X indicates what number it is (nlet_id). This value was set to IN/ON if there can be a user-defined amount of inlets and outlets. We

did this by looping through all the matches found from the regex in Figure 8, and adding the nlet_id to the beginning of each element, and a pipe after each element, and adding this string to a resulting string.

```
let res = '';
if(inlet){
    if(inlet.length !=0){
        res+="|";
    }
    for (let i = 0; i < inlet.length; i++) {
        const match = inlet[i].match(/INLET_+([\s\S]*?);/i);
        // console.log(match);

        if (match ){
            res +=  "I"+ match[1].trim()+ '|';
          }
    }
}
if(outlet){
    if(outlet.length !=0 && !inlet){
        res+="|";
    }
    for (let i = 0; i < outlet.length; i++) {
        const match2 = outlet[i].match(/OUTLET_+([\s\S]*?);/i);

        if ( match2){
            res +="O"+ match2[1].trim()+ '|';
          }
    }
}
```

Figure 10: Code for prepending the nlet_id and appending a pipe to each inlet/outlet element.

This loop is shown in Figure 10, where the resulting string is called "res" and a picture of the resulting string is shown in Figure 11. This string was then added to the index to allow for a fast way to search for a specific tooltip for the inlet and outlet. After this, we had to rebuild the index to update the contents of the search.index file (see Figure 28).

```
|I0 bang rewind clear add add2 set read write print|O0 list|O1 bang|
|I0 set signal|I1 float|O0 bang|O1 bang|
|I0 signal|
|I0 bang|I1 bang|O0 float|
|I0 float bang size nonzero init send receive label
l_pos label_font color pos delta|O0 float|
|I0 float|I1 float|
|I0 float list|I1 float|
|I0 anything|ON anything|
|I0 anything|ON float list symbol pointer bang|
|I0 anything|O1 anything|O0 symbol|
|I0 float bang|I1 bang|O0 bang|
|I0 float bang|O0 float|
|I0 signal|I2 float|I1 signal|O0 signal|O1 signal|

|I0 signal|O0 signal|
|I0 float bang size init number send receive label
l_pos label_font color pos delta|O0 float|
|I0 float list stop|I1 float|I2 float|O0 signal|
|I0 signal bang|O0 float|
|I0 float list bang size scale receive label label_pos
l_font color pos delta|I1 float|O0 float|O1 float|
|I0 float bang size range log init steady receive
label label_pos label_font color pos delta|O0 float|
|I0 signal|O0 signal|
|I0 signal open start stop print|IN signal|
|I0 signal|O0 signal|
```

Figure 11: The console after calling a console.log on res from Figure 10.

From this, we edited the gui_gobj_draw_io() function which is where the inlet and outlet rectangle shape is being created. Setting the contents of the title element required multiple steps. First we had to search the index for the object, as in the first part. This time we search the index with the "obj_text" attribute we created earlier which was set to the object name. Then, using another regular expression, we parsed the string from the index by pipe, storing this into an array. This is shown in Figure 12, where the text highlighted in blue indicates what the regex matches with.

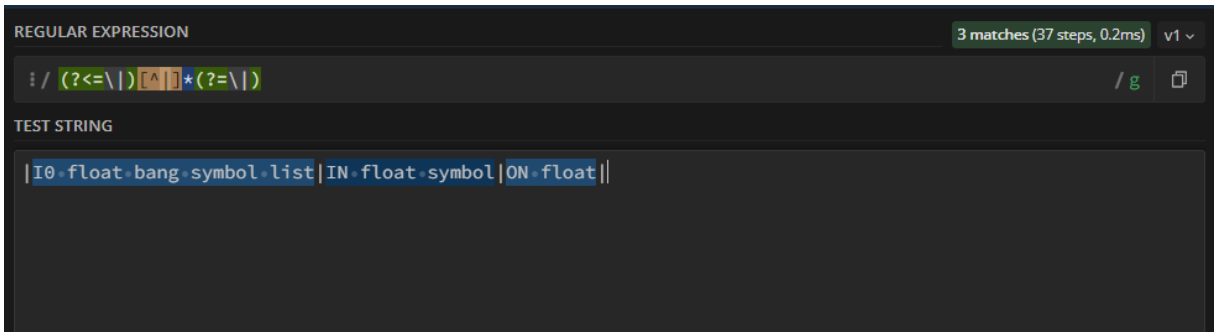

Figure 12: Regex for getting the text between pipes.

We first begin by looping through the array of inlets and outlets from the index and checking for the nlet_id's of the form I followed by a number or O followed by a number. Then we set the title attribute to the corresponding tooltip in the array that we made. After this, if the title attribute is still empty, we check if the inlet/outlet is user-defined. To handle this, we loop through the same array of inlets/outlets, and check for IN/ON, and set the title attribute to everything after the space in the specific element in the array. If the title attribute is still empty after this, then we set the title attribute to "no tooltip available." Then, similar to the first part, we attach the title tag to this newly formed rectangle object.

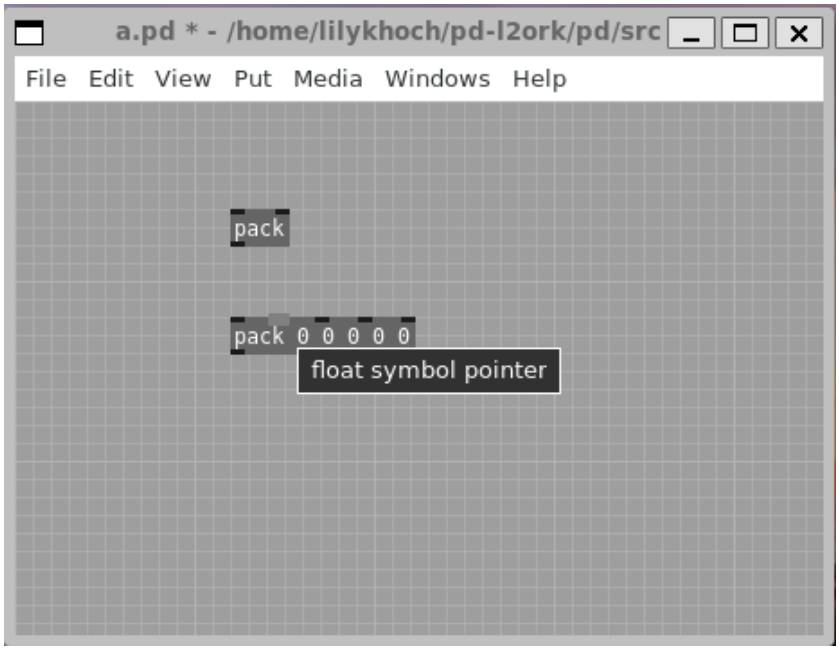

Figure 13: Example of an object with a user defined amount of inlets with tooltips.

In Figure 13, we show an object that can have an N number of inlets. This particular object, pack, allows a user to pack all the inlets into one outlet. In this example, the user is packing the five 0's. The user is currently hovered over the second inlet, shown by the highlighted black rectangle in Figure 13. Similarly to the example in Figure 7, the tooltip for this specific inlet appears.

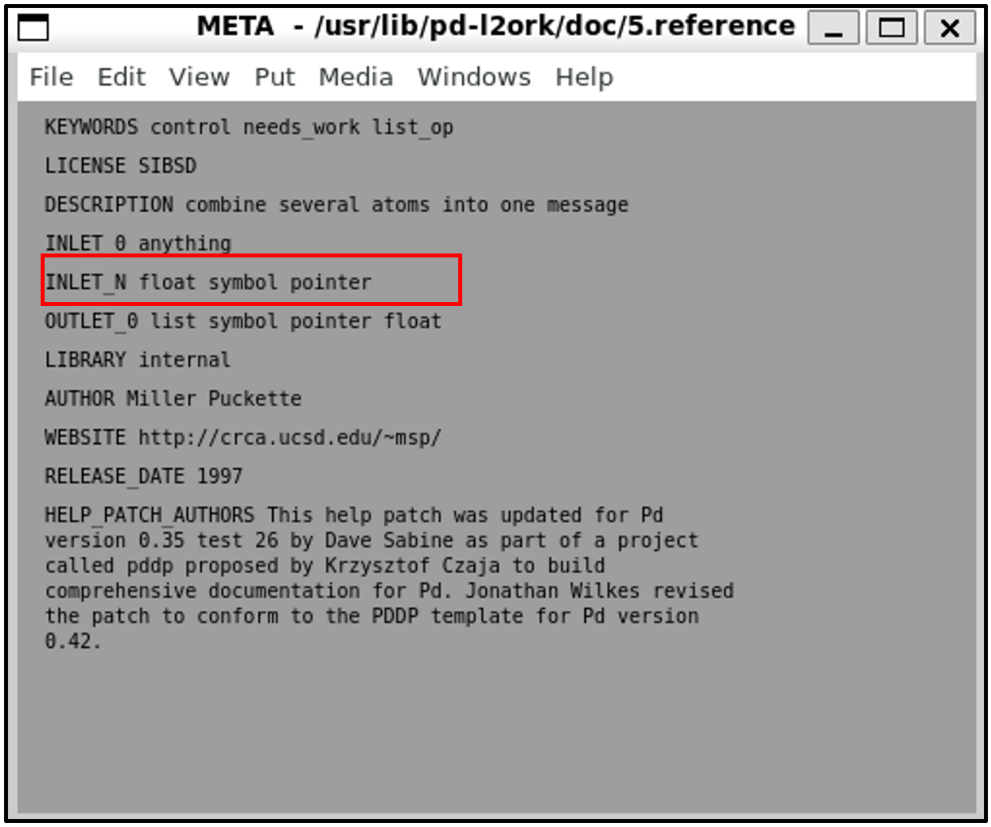

Figure 14: pd help file for the object in Figure 13.

In Figure 14, we confirm that the tooltip for that specific inlet is correct. In this case, since there are a user-defined amount of inlets, the description of the inlet would be INLET_N which is "float symbol pointer." This corresponds to the tooltip that the inlet is getting in Figure 13. Through this process, we were able to implement tooltips for most inlets and outlets. For inlets/outlets that don't have a tooltip, we display "no tooltip available."

As we further progressed through our implementation of tooltips, we ran into multiple issues with the inconsistencies in the naming of objects. For example, the toggle object was searching the index with "tgl" instead of "toggle" which lead to no tooltip being displayed. Most of these

issues were solved by passing the name of the object from the back-end to the front-end so we were getting the correct name for the object for searching. In Figure 15, we call the front-end function gui_gobj_new which is the function for creating new objects. At the end we added two new variables(namebuf and buf) which pass in both the help name and name of the object.

```
char *buf;
int bufsize;
rtext_gettext(y, &buf, &bufsize);

char namebuf[FILENAME_MAX];
gobj_vis_gethelpname(z, &namebuf);

// make a group
text_getrect(&x->te_g, glist, &x1, &y1, &x2, &y2);
gui_vmess("gui_gobj_new", "xxxssiiiiss",
    glist_getcanvas(glist),
    // if it is not toplevel and glist_getcanvas is not gl_owner
    // this means we are drawn 2nd or deeper level down
    glist,
    glist->gl_owner,
    rtext_gettag(y),
    type,
    x1,
    y1,
    glist_istoplevel(glist),
    (pd_class(&x->te_pd) == canvas_class ? 1 : 0),
    namebuf,
    buf
);
```

Figure 15: Back-end call to the front-end for creating a new object.

We also added a new method called gobj_vis_gethelpname (see Figure 16). This allowed us to easily get the helpname for the object wherever the function gui_gobj_new is called on the back-end. This function takes in two arguments a "t_gobj" which is a pointer to the object and a "namebuf" which helps store the resulting name. This function makes use of g_editor.c's canvas_gethelpname function. If the object is a canvas object then the name is copied to the resulting namebuf.

```
extern char *canvas_gethelpname(t_object *ob);

char *gobj_vis_gethelpname(t_gobj *z, char *namebuf) {
    //namebuf = "|null|";

    if (pd_class(&z->g_pd) == canvas_class &&
        canvas_isabstraction((t_canvas *)z))
    {
        t_object *ob = (t_object *)z;
        int ac = binbuf_getnatom(ob->te_binbuf);
        t_atom *av = binbuf_getvec(ob->te_binbuf);
        if (ac < 1)
            return;
        atom_string(av, namebuf, FILENAME_MAX);
    }
    else
    {
        char *obname = (pd_class(&z->g_pd) == canvas_class) ?
            canvas_gethelpname((t_object *)z) :
            class_gethelpname(pd_class(&z->g_pd));
        strncpy(namebuf, obname,
            FILENAME_MAX-1);
        namebuf[FILENAME_MAX-1] = 0;
    }
    post("gobj_vis_gethelpname obname=<%s> namebuf=<%s>", class_gethelpname(pd_class(&z->g_pd)), namebuf);
}
```

Figure 16: gobj_vis_gethelpname function.

Upon further testing of our implementation, we realized we needed to edit the search index again to add another field, "tippathname" (see Figure 17). For objects like K12_output, elasticlunr[8], the search engine used to search through the index, wasn't able to find the correct descriptions for the object. This was due to the backslash in the name that was passed in from the back-end.

```
function init_elasticlunr()
{
    index = elasticlunr();
    index.addField("title");
    index.addField("keywords");
    index.addField("description");
    index.addField("related_objects");
    index.addField("ref_related_objects");
    index.addField("inlets_outlets");
    index.addField("tippathname");
    index.setRef("id");
    return index;
}
```

Figure 17: Adding "tippathname" as a field in the index.

As a result we needed to added the "tippathname", which is the path to the help file with all the backslashes replaced with spaces (see Figure 18). We added this field after the inlets/outlets field in the search index.

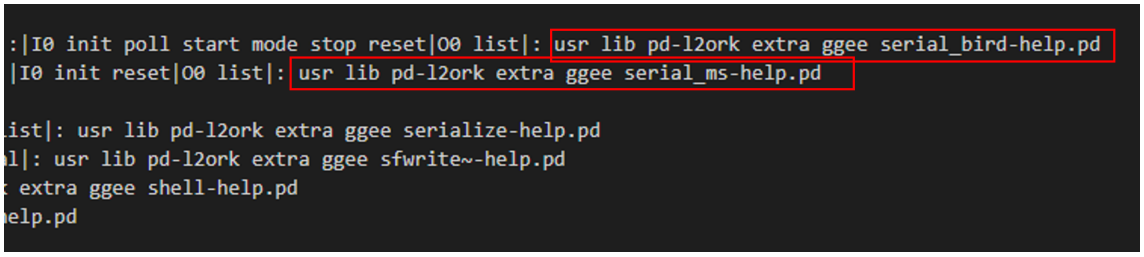

Figure 18: Resulting search index with "tipppathname."

With this new field, we were able to edit the way we search for the objects with the tippathname. For example, K12_output would search the index for "K12 output" and find the relating description there (see Figure 19). Minor changes to the inlet/outlet searching were edited to reflect the same changes.

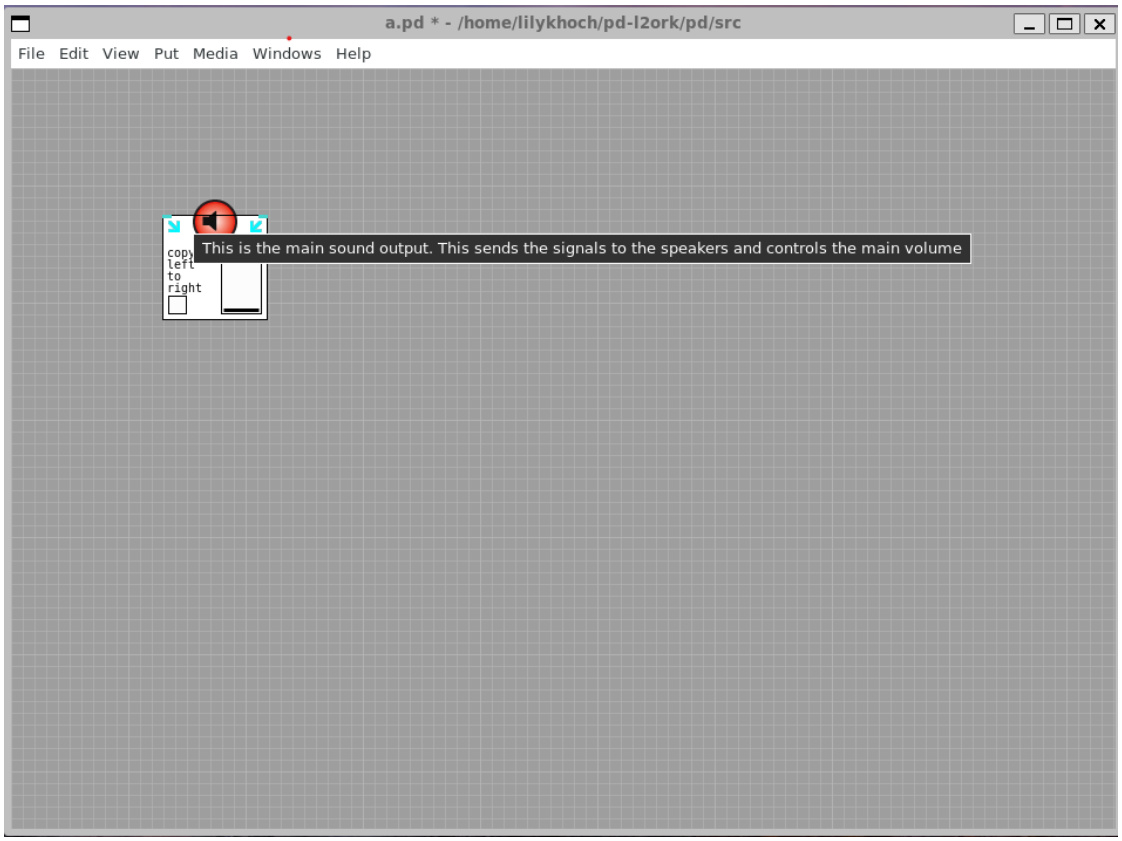

Figure 19: K12/output tooltip.

5.2 Web Browser Button Implementation

The first part of our task was research-focused. Knowing little to nothing about Emscripten[2] and WebAssembly, we looked into how the Emscripten compiler, emcc [9], did its job of compiling C code into something a web browser could use. We also learned about some of the limitations of Emscripten, which we will explain later in more detail.

Part of this research was looking through examples of people using Emscripten for similar projects. Zack Lee was part of a larger project to compile Purr-Data[4], which is a fork of Pd-L2Ork, with Emscripten. You can see what he created in Figure 20. In addition, we emailed Zack Lee to gain further insight into how he managed this code transformation. Our team received valuable insights on what steps we needed to take next, from him. We also examined an example of someone compiling the original Pd software in Emscripten[9]. From these examples we gained a basic understanding of what was required.

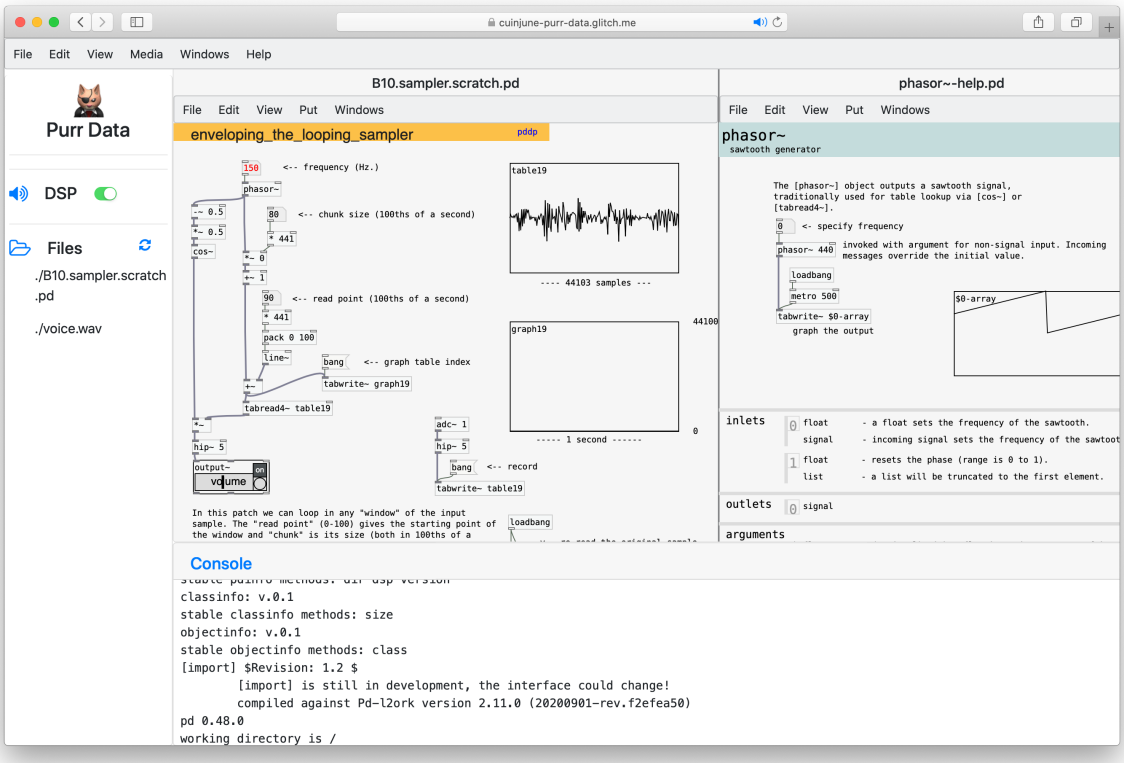

Figure 20: Purr-Data can not only display patches in a web browser, but inserts the entire pd editor into the web so you can create patches in the browser, not just view and interact with them.

We then tried to download and run Zack Lee's Purr-Data (Emscripten branch) and his PdWebParty, which is a front-end extension that takes files generated by Purr-Data's Emscripten compilation process and runs them to display on a browser, as shown in Figure 21. We were able to successfully run both of these after some trial and error, confirming that the process was indeed possible and we could draw from these sources as working examples.

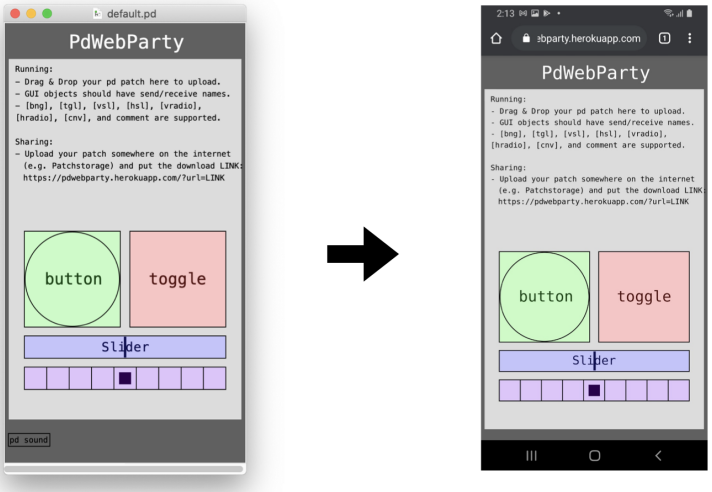

Figure 21: PdWebParty takes a sample patch pre-compiled with Emscripten and displays it in a browser using a simple front-end server.

From there, we dove right in to downloading the Emscripten compiler and seeing what compiled and what didn't in Pd-L2Ork. We were able to copy over some of the Makefiles from Zack Lee's Purr-Data example to get started on compilation. We pretty quickly ran into errors, however, and unfortunately solving these errors would take up the majority of our time.

The cause of most of these errors was incompatible C code with Emscripten [10]. One thing that Emscripten cannot handle is casting function pointers and then calling those pointers. An example of this can be seen in Figure 22. Another thing Emscripten can struggle with is passing structures by value as opposed to passing a pointer. These errors, along with some others, considerably slowed down our progress in compiling the entirety of Pd-L2Ork.

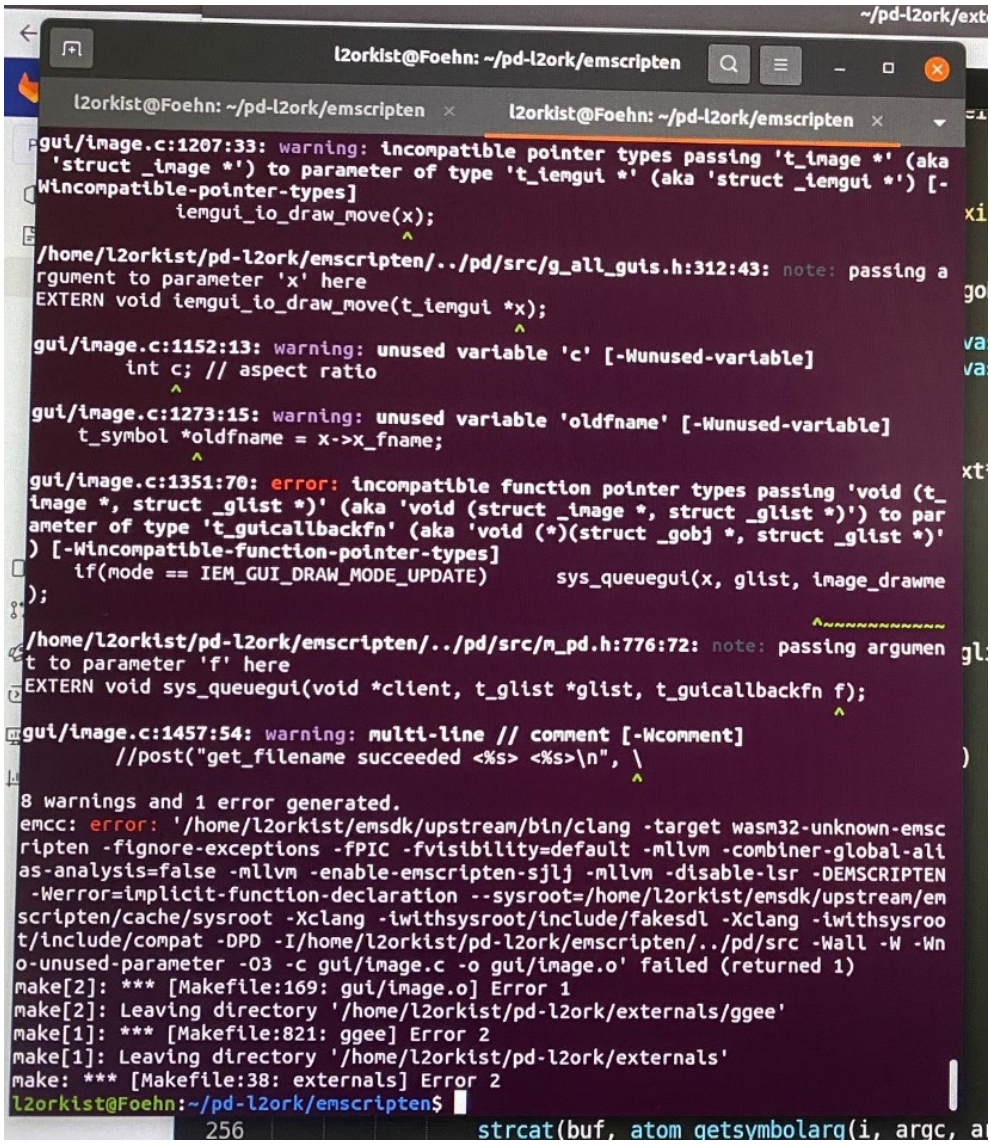

Figure 22: An example of a function pointer error that we encountered during compilation.

The main cause of the slowdown was compiling the externals folder of the project, which contained many problems similar to the ones stated above and unfamiliar ones as well. We left most of the fixing of the complicated errors to our client since his knowledge of the codebase was far more in-depth than ours, but we were able to fix some of the errors just by observing what Zack Lee had changed in Purr-Data. In the cases where the files between Purr-Data and Pd-L2Ork were identical or nearly so, we were able to directly port over his changes into Pd-L2Ork and in most cases, that solved the problem. However, in some cases the files either did

not exist in Purr-Data or they were different to the point where Zack Lee's changes were either indiscernible or did not apply to our version of the file. In this case we had to do more digging into the error and tried to match the error with an incompatibility case from the aforementioned Emscripten article. Sometimes we were able to locate the problem after digging, or our client was able to code a fix, but sometimes the problems were simply too persistent. If we ever spent more than a few days trying to debug a problem compiling to no avail, we skipped over compiling that specific external and moved on to the next one. Developers can learn more about these unfinished externals and how to fix them in the Future Work Section.

Another concern that we had to deal with while compiling the externals was edits Zack Lee made to the Makefile. For certain externals, Zack Lee had made changes not only to the external files themselves, but also to the way they were made, either to avoid compiling certain files that were problematic or to compile them with certain settings, often special flags. An example of this is shown in Figure 23. All these changes had to be manually tracked down and implemented in the Pd-L2Ork "make" process in order to solve more compilation errors.

```
386 #-------------------------------------------------------------------------------#
387 # BSAYLOR
388 bsaylor:
389         make -C $(externals_src)/bsaylor PD_PATH=$(pd_src) CFLAGS="$(CFLAGS) -fno-strict-aliasing" $(call set_em_flags)
390
391 bsaylor_install:
392         make -C $(externals_src)/bsaylor \
393                 DESTDIR="$(DESTDIR)" objectsdir="$(objectsdir)" $(call set_em_flags) install
394         $(call if_emscripten, -rm -f $(DESTDIR)$(objectsdir)/bsaylor/pvoc~* $(DESTDIR)$(objectsdir)/bsaylor/partconv~*)
395
396 bsaylor_clean:
397         make -C $(externals_src)/bsaylor clean
398
399
400 #-------------------------------------------------------------------------------#
```

Figure 23: Zack Lee systematically removing a problematic file so that emcc does not attempt to compile it and throw an error.

After finally finishing this part of the task, we were able to compile the externals with emcc. Our next task was to compile the main library and run the build folder's Makefile so that the JavaScript files could be generated. In order to do this we needed to copy some more code over from Zack Lee's implementation in Purr-Data. This included some C++ files that define libpd wrapper functions that can be accessed by the front-end. It also includes the libpd folder that contains a bunch of helper files. The next logical step would be to implement logic to generate the files needed to import into PdWebParty for any patch on the fly so that we have something

tangible to test and that future developers can work with.

5.3 Functionality of What Team Develops

In Table 1, we list the steps to complete each main task that is carried out by both the tooltips and web browser teams and describe the implementation of the service. Included is implementation-specific information such as the specific input and output files we use for each specific task and the specific environments that we use. Each task is given an "ID" which is matched to the corresponding relationship in our design graphs shown in Figures 1 and 2.

Service ID	Service Name	Input file name(s)	Input file IDs	Output file name	Output file ID	Libraries; Functions; Environments
1A: Data Storage	Data Storage: Create data storage mechanism to hold patch metadata	g_text.c	1			Depends on preference, either Ubuntu and Sublime text[11] or macOS and VSCode[12]
1B: Data Storage	Data Storage: Parse through metadata to get the description	g_text.c	1			Depends on preference, either Ubuntu and Sublime text[11] or macOS and VSCode[12]
1C: Display tooltips	Display tooltips: Check how many inlets and outlets each object has (maybe checking with the separators) and add info and title tags			pdgui.js	2	Depends on preference, either Ubuntu and Sublime text[11] or macOS and VSCode[12]

1D: Test tooltips solution	Test tooltips solution: Make sure tooltips work for variable inlet/outlet objects					Depends on preference, either Ubuntu and Sublime text[11] or macOS and VSCode[12]
1E: Test tooltips solution:	Test tooltips solution: Merge implementation with main code base. Test again once merged	N/A		N/A		Pd-L2Ork environment
2A	Bring copy-able code directly over into our implementation	N/A		Likely this will be most of the front-end files, and all .HTML files	2, 5, 6, 7	N/A
2B	Write new code needed to implement web browser functionality on Pd-L2Ork	Pdgui.js, all existing .HTML files	2, 7	A new "version" of pdgui.js that's made for web browser, using all the same components but in a different format	2.1	WebAssembly, HTML5 GUI framework

2C	Merge copy-able and newly written code to create the exporting system	pdgui.js, g_text.c	1, 2	Same as above, slightly modified elements for the web version	2.1, 7.1	WebAssembly, HTML5 GUI framework, Emscripten
2D	Link front-end web page to back-end engine	pdgui.js, all portaudio.c files	2, 3	A JS file that can communicate between the front-end and portaudio engine (jstoengine.js)	4	PortAudio Library, will likely need to support multiple OS environments (Mac OS, Windows, Linux)

Table 1: Functionality

6 Plan

After meeting with our client Dr. Bukvic initially, we scheduled weekly meetings with him to ensure steady progress on the project. We met each week in DISIS, the Digital Interactive Sound and Intermedia Studio, and checked out Linux laptops per our client's suggestion to work on Pd-L2Ork. Our team initially started out working together and familiarizing ourselves with the Pd-L2Ork code base. After some initial difficulties installing Pd-L2Ork, we were ready to start working. Once we had a better idea of the tasks at hand, we decided to split up into two groups since our client had two different tasks for us to complete. Lily and Sahar chose to work on the tooltips functionality, while Drew and Ethan chose to work on integrating the web browser button into the main branch of Pd-L2Ork. We also came up with a timeline, shown below, to keep the team on track to finish by the end of the semester.

February

1. Familiarize group with code base
2. Get in touch with Zack Lee (for embedding web browser part)
3. Start implementation of tooltips/web browser button

March

1. Continue implementation of tooltips/web browser button
2. Testing/debugging

April

1. Finalize implementation of tooltips/web browser button
2. Merge code with main repository

7 Testing

7.1 Tooltips

Since the edits for tooltips were mainly done on the front-end side, testing was easier. To test, we had to make our edits to the pdgui.js file, and compile again by running "make" and copying the pdgui.js file to the bin folder, and then run Pd-L2Ork. This allowed us to see the changes that we made and allowed us to easily debug our code. We were able to check the correctness of our code by verifying that the tooltips correspond to the correct object. We also made use of console.log and post statements to print variables that we needed to see, making it easier to figure out possible errors in our code. There were some issues when testing for all the cases for the inlets/outlets part because we were unsure if we had full coverage for all cases.

7.2 Web browser button

There is not a currently completed version of the web browser button, so no testing of any significance was done up to this point.

8 User Manual

8.1 Installing Pd-L2Ork

Users will first need to download and install Pd-L2Ork, which can be found at: https://github.com/pd-l2ork/pd-l2ork/. There are build instructions for Windows, MacOS, and Linux, with Linux being the recommended operating system of choice as shown in Figure 24. Alternatively, users can go to http://l2ork.music.vt.edu/main/make-your-own-l2ork/software/ and follow the instructions for the "Burrito Supreme" version of Pd-L2Ork.

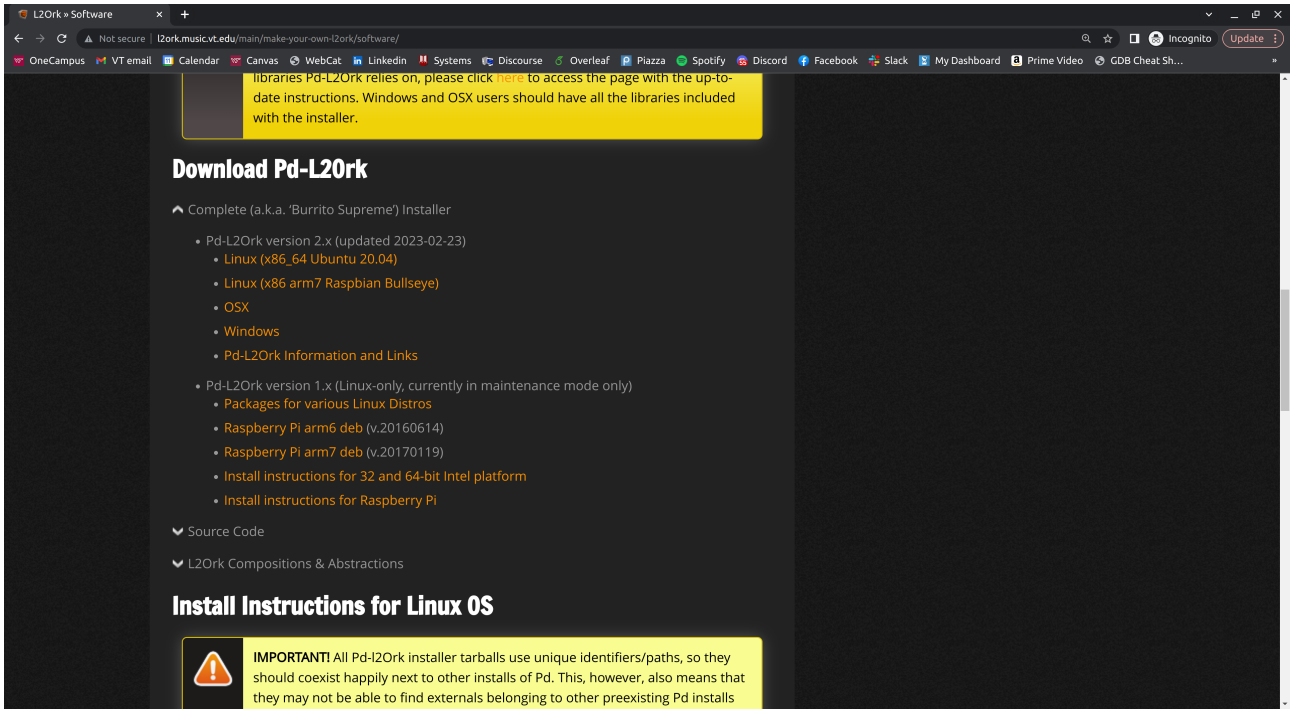

Figure 24: Download options for installing Pd-L2Ork on different operating systems.

8.2 Using Tooltips

The idea behind the tooltips is to give some basic information about an object and its inlets and outlets to the user. This should make things easier for the user during patch creation. The tip for an object will appear by just hovering over the object.

Objects may have one or more inlets and outlets which appear on the top part and bottom part of the object, respectively, as a black rectangle. The number of the inlets and outlets can

also depend on the parameters that the user passes into the object. By simply hovering over the inlets and outlets, the tooltip corresponding to that specific inlet/outlet will show up (see Figure 25).

Figure 25: A user hovering over a print object.

In both cases, there is a possibility that a tooltip is incorrect or there isn't a tooltip available. These will be shown as "fixme" or "no tooltip available" when hovered over (see Figure 26).

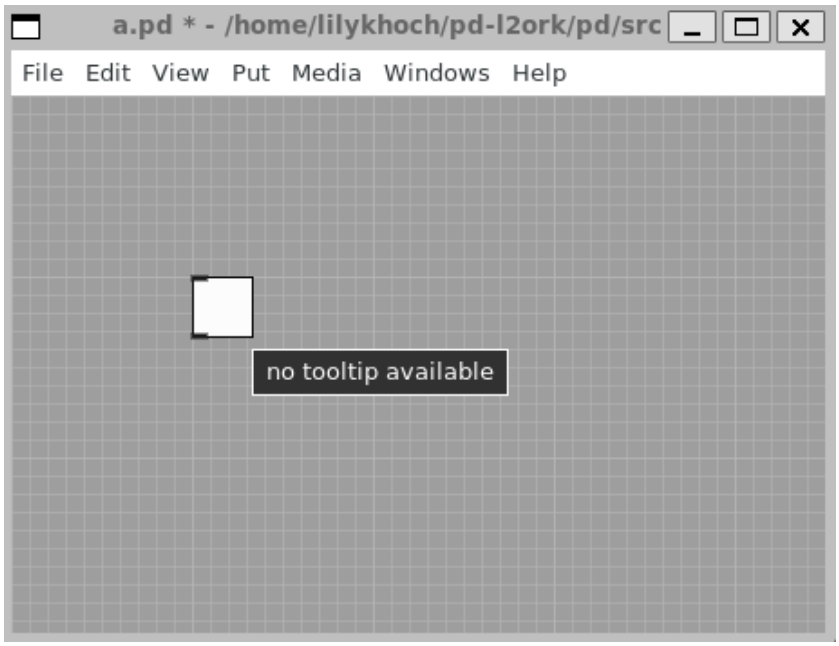

Figure 26: An object that does not have a tooltip available.

For certain objects, there are specific places where a you must hover over in order for a tooltip to display. The best spot to hover to display the tooltips is around the edge of the object. K12 objects have this issue since the object is made up of multiple objects, but for objects like toggle and numbox, hovering over any part of the object will suffice. See Figure 27.

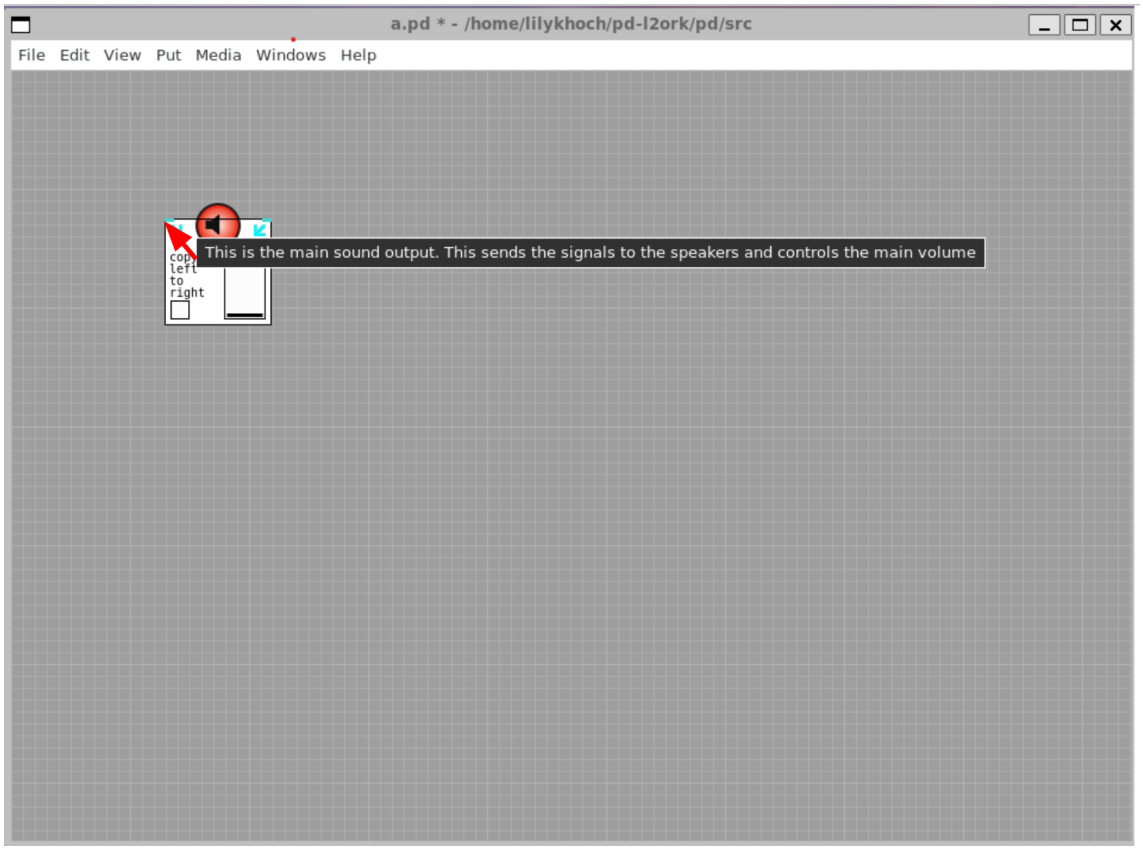

Figure 27: A K12 object and getting the tooltip by hovering over the edges only.

8.3 Using Web Browser Button

There is not a currently completed version of the web browser button, so users who want to see how they can produce one can go to the Future Work section.

9 Developer's Manual

Pd-L2Ork is a very large program, and our team didn't touch the majority of its files. Even within the files we do edit, we often only edit a small portion of the file. This is especially true for the tooltips team, so for the future developers we will focus only on the relevant files and folders we worked in.

9.1 Tooltips

9.1.1 Functions and Tools

To further expand the tooltips implementation, we recommend using a Linux system or some type of Linux environment. If you have a Windows system, the Windows subsystem for Linux (WSL) [13] is a great tool to use. We would start by taking a look at the following files to fully understand the scope of the implementation. There are three main files that we focus on, the pdgui.js, the g_text.c, and the g_all_guis.c files.

- pdgui.js: This JavaScript file houses the main tasks for front-end development such as object creation, index building, canvas creation, etc. All of the parsing and displaying of the tooltips happens in this file. In the function add_doc_details_to_index(), we parse the information from metaData to the index.search file, Figure 28. As seen in Figure 28, inlets and outlets information are at the end of each line related to each object, and each inlet and outlet is separated by a pipe(|). Other information is separated by a colon(:). Then in gui_gobj_new() function, will get the tips based on the name of an object or the inlets and outlets number. A parameter was added to the gui_gobj_new() function in order to get the name of an object. In the gui_gobj_draw_io() function, the inlets and outlets are made. In this function the rectangle for each inlet and outlet is created, and the corresponding tooltip for it is attached as a title tag.

- g_text.c: Since the gui_gobj_new() function was modified in pdgui.js, some changes needed to be made to where this function is being called in this file. This is a back-end file that passes the name of the object to the front-end.

- g_all_guis.c: This file is responsible for more of the back-end communication with the front-end.

- search.index(see Figure 28): Pd-L2Ork utilizes Elasticlunr.js, "a lightweight full-text search engine developed in JavaScript for browser search and offline search." [8] The search.index file stores information on every object in Pd-L2Ork which allows for easy searching when adding a tooltip to each object based on the object name. We added 2 more fields to each object: inlets_outlets and the "tippathname" which stored information on the inlets/outlets and the path name for each object help file.

It is important to keep in mind that the name of some objects might not match the corresponding name of that object in the index file. In order to search for an object in the index file, the "tippathname" field can be used, which is based on its corresponding help file name in the index file (see Figure 18).

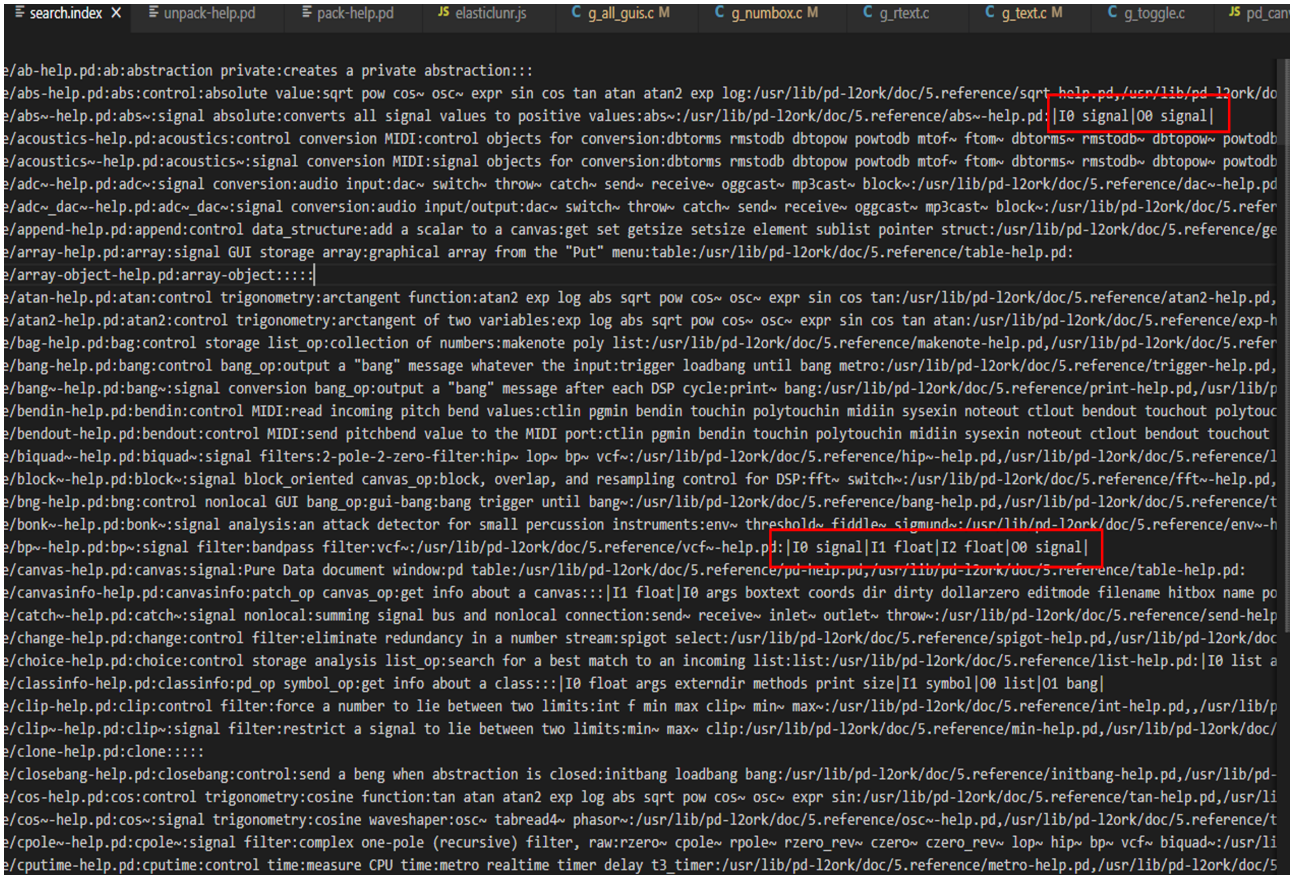

Figure 28: The search.index file with the new changes.

9.1.2 Relevant Files

Locations of important files in the repository:

- pdgui.js: /pd-l2ork/pd/nw/pdgui.js
- g_text.c: /pd-l2ork/pd/src/g_text.c
- g_all_guis.c: /pd-l2ork/pd/src/g_all_guis.c
- search.index (see Figure 28): /home/user/.pd-l2ork/search.index
- g_magicglass.c:
- g_mycanvas.c:

Externals (/pd-l2ork/externals):

- ggee/gui/w_envgen.h
- miXed/cyclone/sickle/Scope.c
- moonlib/image.c
- pddp/helplink.c
- tof/src/imagebang.c
- unauthorized/pianoroll.c
- ggee/gui/image.c

All the help files corresponding to all the objects can be found at:

- /usr/lib/pd-l2ork/doc/5.reference

9.2 Web Browser Button

9.2.1 Relevant Tools

To continue working on the web browser portion, a few things are needed:

1. The base version of Pd-L2Ork: https://github.com/pd-l2ork/pd-l2ork/
2. Purr-Data: https://git.purrdata.net/jwilkes/purr-data/-/tree/emscripten/
3. PdWebParty: https://github.com/cuinjune/PdWebParty
4. empd: https://mathr.co.uk/empd/
5. Pd-L2Ork emscripten branch: https://github.com/pd-l2ork/pd-l2ork/tree/emscripten
6. Emscripten SDK: https://github.com/emscripten-core/emsdk

Each of these needs to be installed on your local machine (except empd). As mentioned previously, we highly recommend using a Linux environment, either by borrowing a laptop from Virginia Tech or through using a virtual Linux machine. Pd-L2Ork is, after all, made primarily for Linux machines, and using a different operating system just doesn't work well based on our experience. The instructions for downloading and installing each of these tools can be found on their respective pages.

9.2.2 Relevant Folders/Files

The web browser team spent most of their time in the following folders:

1. Pd-L2Ork-Emscripten/externals
2. Pd-L2Ork-Emscripten/emscripten
3. Pd-L2Ork-Emscripten/libpd
4. Pd-L2Ork-Emscripten/pd/src

The web browser team initially spent a lot of time in the externals folder correcting various errors that come up when compiling with Emscripten. We were able to get this portion working, so future developers shouldn't have to deal with this portion.

The second major folder we worked in is the Emscripten folder. Here we added the build and src folders and successfully ported over the Makefile from Purr-Data, along with some changes, to get it to work.

In the third folder, libpd, we added many necessary files from Purr-Data to get the Makefile to work. Future developers shouldn't have to work on this folder either. The final folder, pd/src, is the one the web browser team left off. Many changes were made here in response to the errors we were getting in PdWebParty. Here, we are carefully looking at the commit history of Purr-Data to track and port the necessary changes to Pd-L2Ork.

For future teams, the next step is to modify the build script to include both the regular Pd-L2Ork and Emscripten portions. The build script in question is the "tar-em-up.sh" file in the folder 'l2ork_addons'. When doing this, make sure to look at Purr-Data's build script because the structure should be similar. Future teams should make sure to carefully look at the commit history of Purr-Data whenever making changes to code, as that often provides valuable insight.

9.2.3 Running in PdWebParty

To run in PdWebParty, copy the files main.js, main.wasm, main.html, and main.data generated from a successful build of Pd-L2Ork (with Emscripten) in the Pd-L2Ork-Emscripten/emscripten/build folder to PdWebParty's PdWebParty/public/emscripten folder. After that, do "npm start" in PdWebParty's root directory and then open up the link in a browser that supports dev tools like Google Chrome[14]. The result should look like Figure 29, when opened with Google Chrome's dev tools.

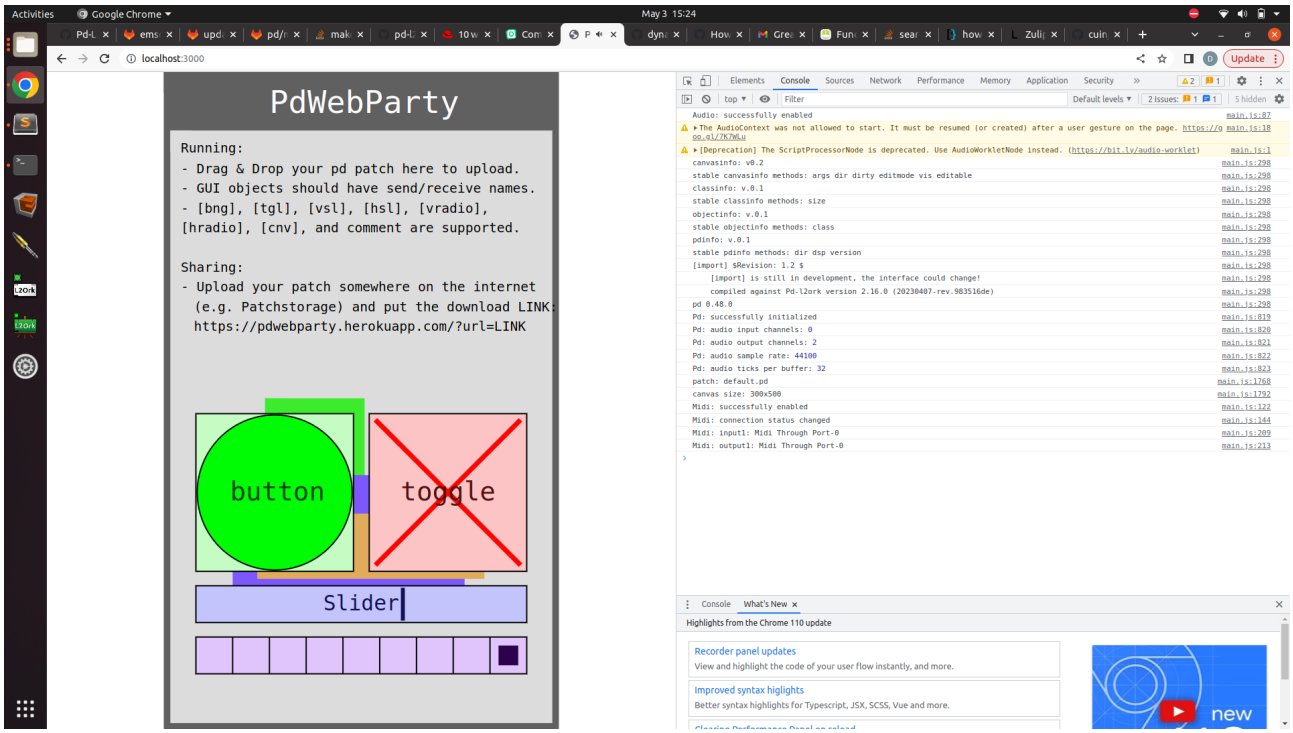

Figure 29: Example of what PdWebParty looks like after copying over the main.* files.

Note: If you try running in PdWebParty and get an "FS Error" at runtime, as shown in Figure 30, then you need to change a few lines in PdWebParty's public/js/main.js to resolve the error.

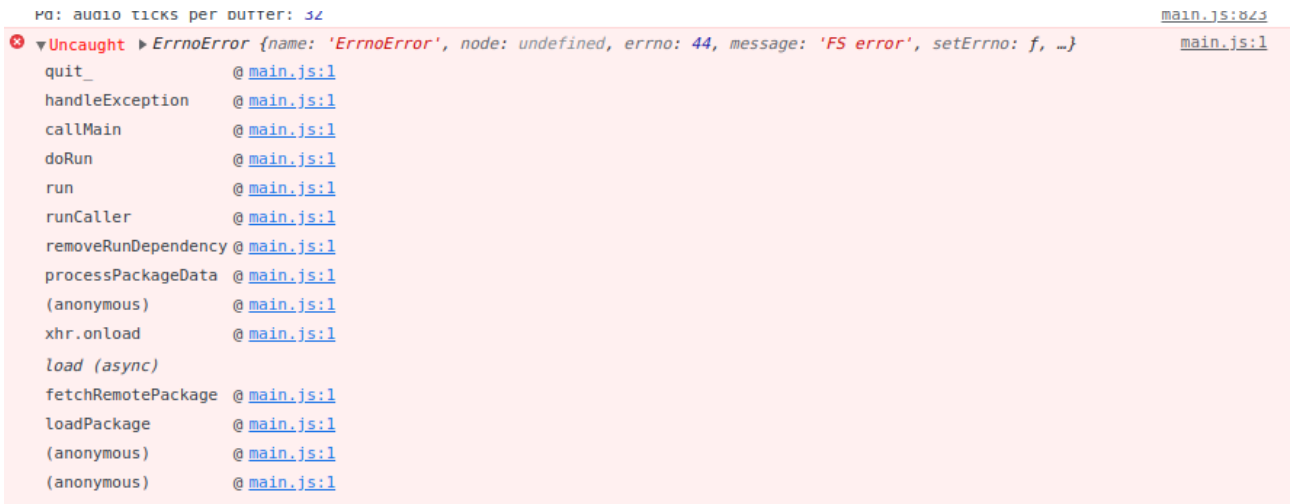

Figure 30: The FS error you might get if code in PdWebParty needs to be changed.

Look for the lines declaring "var helpPath" and "var extPath", shown in Figure 31, and change the first folder to the name of the folder wherever your version of Pd-L2Ork is built. For the current version of the Pd-L2Ork Emscripten Repository, this is "pd-l2ork-web". So the two lines should now look like:

```
824
825             // add internals/externals help/search paths
826             var helpPath = "pd-l2ork-web/doc/5.reference";  // Changed from "purr-data/doc/5.reference"
827             var extPath = "pd-l2ork-web/extra";             // Changed from "purr-data/extra"
828             pd.addToHelpPath(helpPath);
829             pd.addToSearchPath(extPath);
```

Figure 31: The lines to change in public/js/main.js and how you would change them in the Emscripten Pd-L2Ork Repo.

10 Lessons Learned

10.1 Timeline

February

1. Familiarize group with code base
2. Get in touch with Zack Lee (for embedding web browser part)
3. Start implementation of tooltips/web browser button

March

1. Continue implementation of tooltips/web browser button
2. Testing/debugging

April

1. Finalize implementation of tooltips/web browser button
2. Merge code with main repository

10.2 Problems

Our team had some issues downloading and installing some versions of Pd-L2Ork to begin with. Some of the team members chose to develop the software through their own personal computers while others chose to borrow a Virginia Tech-owned Linux[15] Laptop. One team member initially started developing on their own laptop running on MacOS[16] but had installation troubles due to the unique nature of their computer's environment and eventually switched to borrowing a Linux Laptop.

For the tooltips team, we ran into some issues in the beginning with adapting to the large code base. It took a long time to fully understand the code and what it does. Another challenge we faced during the implementation of the tooltips for the objects was the inconsistencies in object name and the object name in the index. For example, the toggle object name is "tgl", however in the index the name for it is "toggle." This lead to an issue when we were searching the index for the object with the actual object name "tgl" instead of "toggle" so the

search results were null. We also faced issues with the tooltips for the inlets/outlets when we were implementing the part that attaches the correct tooltip to the corresponding inlet/outlet. Another issue we faced was the naming of objects that included backslashes. This lead to numerous issues with searching that took a lot of our time to figure out.

For the web browser button team, we ran into numerous problems throughout the development process. Firstly, neither one of us had any prior knowledge of Emscripten [2] or attempting to run C code in a web setting. We also had limited knowledge of the Linux [15] operating system. The next major problem we ran into was that the current version of Pd-L2Ork was not compatible with Emscripten, and caused the compiler (emcc) to throw a bunch of errors that required fixing. The last big problem we encountered was editing the Pd-L2Ork build script so that the full Pd-L2Ork plus all of the required web files could be generated with one run of the build script.

10.3 Solutions

For the tooltips team, we overcame our issues by consistently communicating with Dr. Bukvic about any questions we had about implementation ideas or about specific objects themselves. Since most of our issues stemmed from the lack of experience with Pd-L2Ork and how users use each object of the software, Dr. Bukvic was able to provide us with examples of specific processes that users make when creating an object. This helped us better implement a solution.

For the web browser button team, we overcame our lack of knowledge with Emscripten and Linux by doing a bunch of research on how Emscripten went about compiling C code into WebAssembly and Makefile structures before starting to make code edits. In order to solve the emcc compiling issues, we closely analyzed Zack Lee's Emscripten branch of Purr-Data [4] to see if he had made changes to the files we were getting errors from, and we then made those changes in an attempt to fix the compiling errors. This worked in some cases. However, in some cases, the disparity between the two files was so great that the changes he made to fix emcc errors either were not evident or did not exist. In those cases, we closely examined the error message and attempted to find the problem ourselves, while also simultaneously reaching out to Dr. Bukvic to see if he could make sense of the error, since we knew that he knew the code base much better than we did. Any errors that we could not fix he was either able to help us with or told us to bypass so we did not get bogged down in one error.

10.4 Future Work

For the tooltips team, we have a working implementation for most objects. The next steps would be to check for the cases where there aren't any tooltips for that specific object and attach a tooltip to it. In our current implementation, we handle these cases by setting the tooltip to "no tooltip available". A few of the externals listed in Section 9.1.2 were not displaying any tooltips. This is possibly caused by the object not having a description or an object not having an associated help file. These objects will also display "no tooltip available." More user testing and collaboration with Dr. Bukvic will be needed to ensure that the system handles all cases.

For the web browser team, we wished we could've gotten farther in the compilation process throughout the semester and that it would not have been as time-consuming as it was. The immediate next step for the team would be to automate the build script to put all the necessary files into the correct place in the Pd-L2Ork file structure, and then to create logic to pull the necessary files out of the build folder to create the exported patch on the web page. Our team hasn't had the time to set up a web server to host this on, but that would be another future step worth considering for any teams that take up this work, in lieu of using an edited version of PdWebParty. Another optional step for the next team to work on is to fix the bugs causing some of the externals not to compile. Currently the compile process is skipping over some externals in order to allow the process to complete. The externals that are currently skipped over are as follows:

- iem_ambi
- iem_bin_ambi
- iemguts
- iem_adaptfilt
- iem_delay
- iem_roomsim
- iem_spec2

If you go to the externals folder and look at the Makefile there, you can find under the "# ALL" header the build targets for emscripten compilation. To debug these externals, uncomment the

original build target list and comment out the modified one, and then debug the errors that appear when compiling.

11 Acknowledgements

Special thanks to Professor Ico Bukvic[6], our client, for helping us discover an incredible computer music tool in Pd-L2Ork and allowing us to assist him in the innovation of his software. He can be contacted at ico@vt.edu.

Thank you to Zack Lee for providing us with valuable insight on the web browser side of our project and pointing us in the right direction.

And as always, thank you to Dr. Edward Fox for allowing us to work on this project and get experience with a client.

12 References

References

[1] D. Moreno, H. Kwok, N. Nguyen, and T. Johnson, "Pd-L2Ork," in *Pd-L2Ork*, 2023. Virginia Tech CS4624 Fall 2022 Team Term Project, http://hdl.handle.net/10919/112908, last accessed on 03/02/23.

[2] Emscripten Contributors, "Emscripten," 2023. https://emscripten.org, last accessed on 02/24/23.

[3] Z. Lee, "PdWebParty Github," 2023. https://github.com/cuinjune/PdWebParty, last accessed on 02/28/23.

[4] J. W. Zack Lee, "Purr-data Emscripten Branch Github," 2023. https://git.purrdata.net/jwilkes/purr-data/-/tree/emscripten/, last accessed on 02/28/23.

[5] I. I. Bukvic, "l2ork," 2023. Virginia Tech Department of Music web page, https://l2ork.music.vt.edu/main/what-is-l2ork/, last accessed on 02/27/23.

[6] I. I. Bukvic, "Virginia Tech Music Faculty page for Professor Bukvic," *Virginia Tech School of Performing Arts*, 2023. https://sopa.vt.edu/faculty_staff/music-faculty/comp-and-creative-tech-faculty/ivica-ico-bukvic.html, last accessed on 03/02/23.

[7] Scott Chacon, Beb Straub, "Git," 2014. https://git-scm.com, last accessed on 04/11/23.

[8] Wei Song, "Elasticlunr.js," 2015. http://elasticlunr.com/, last accessed on 05/3/23.

[9] Empd Contributors, "Empd," 2023. https://mathr.co.uk/empd/#how, last accessed on 03/15/23.

[10] Emscripten Contributors, "Function Pointer Issues," 2023. https://emscripten.org/docs/porting/guidelines/function_pointer_issues.html, last accessed on 02/24/23.

[11] Sublime Text Contributors, "Sublime Text," 2023. https://www.sublimetext.com/, last accessed on 04/09/23.

[12] VSCode Contributors, "VSCode," 2023. https://code.visualstudio.com/, last accessed on 04/09/23.

[13] Microsoft, "WSL," 2023. https://learn.microsoft.com/en-us/windows/wsl/install, last accessed on 05/3/23.

[14] Google Inc., "Google Chrome," 2023. https://www.google.com/chrome/, last accessed on 05/3/23.

[15] Linux Contributors, "Linux," 2023. https://www.linux.org, last accessed on 04/11/23.

[16] Apple, "What is MacOS?," 2023. https://www.apple.com/za/macos/what-is/, last accessed on 04/11/23.

13 Appendix A: Methodology Assignment

13.1 Types of users

1. New user, like a student who isn't familiar with Pd-L2Ork or programming in general, who would probably benefit more from the tooltips
2. "Professional" user, like Prof. Bukvic and the teams he makes music with, who could also benefit from tooltips but are more likely to use the web browser functionality

13.2 User Goals

- Learning about the system: Have easy ways to navigate the software and quickly learn about how each patch works.
- Teaching about the system: Have effective ways to teach a large amount of students about how the software works in an efficient and easy-to-use manner.

13.3 User Goal: Learning about the system

- Learn about audio flow
  - Learn through patches that change audio through simple and intuitive ways
  - Connect different patches to observe how audio flows between them
- Learn about system functionality
  - Learn about how patches work
  - Learn about patch cords
  - Learn about what you can accomplish with Pd-L2Ork
    - Explore documentation and help menus
- Learn about individual patches
  - Learn about patch abstraction
  - Learn about how patches transfer information through inlets and outlets
  - Connect patches together with cords to see how they interact

13.4 User Goal: Teaching about the system

- Teach about the system to many students at once
  - Utilize DISIS or other similar rooms to showcase PD-L2Ork to many students at a time
  - Make Pd-L2Ork easily portable to many devices
    - Create an easy way to reproduce the same patch in many devices
- Make the system easy for students to interact with
  - Create a way to view the inputs and outputs of each patch easily and intuitively
  - Make patches simple enough to understand complex processes easily
- Make the system easy to compartmentalize for teaching
  - Utilize the existing K12 mode to teach concepts important to learn in earlier stages of education
  - Come up with a way to quickly share patches and their effects with students