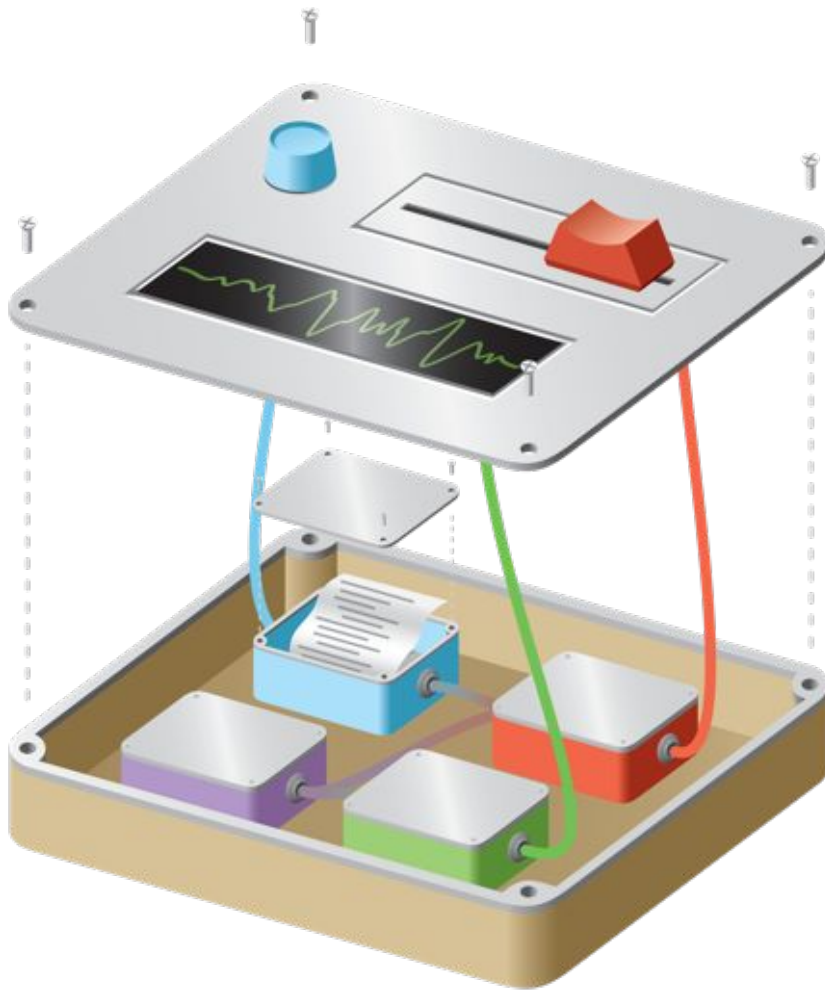


MEEMOO: HACKABLE WEB APP FRAMEWORK

FORREST OLIPHANT



Media Lab Helsinki
Aalto University
March 2012 – version 1.0

<i>Author</i> Michael Forrest Dysart Oliphant		<i>Year of publication</i> 2012
<i>Department</i> Media	<i>Degree programme</i> Master of Arts, New Media	
<i>Title</i> Meemoo: Hackable Web App Framework		
<i>Type of work</i> Project and performance	<i>Language</i> English	<i>Number of pages</i> 60
<i>Abstract</i> <p>Meemoo is a modular web app framework with a focus on design for hackability. People can create and modify new media tools with this framework in a web browser without writing code. Tools are made by wiring together pre-built modules. Modules are created with web standards.</p> <p>This thesis consists of a software project and a performance. The software project is Meemoo[1], which is a framework for hackable web apps. The performance consisted of live animation[2] made with the Meemoo framework.</p> <p>The main objectives of this project:</p> <ul style="list-style-type: none"> • Design a modular dataflow visual programming framework using web technologies. • The framework should afford non-coders the ability to modify creative web apps by configuring wires that represent how modules communicate. • Apps created with the framework should have source code that is easy to read and share. • Web coders should be able to extend the framework by creating modules using web standards. There should be a simple syntax to define the inputs and outputs of a module. <p>---</p> <p>[1] Project page, demos, and source code: http://meemoo.org/ [2] Performance documentation: http://youtu.be/T_tCyYGLWKM</p>		
<i>Keywords</i> web, html, javascript, framework, design, modular, hackable, open source		

ACKNOWLEDGEMENTS

Helsinki, March 23, 2012.

Two years ago I was a beach bum in Dunedin, New Zealand, happy but thinking about the next adventure. Knowing nothing about Finland, I applied to Media Lab Helsinki. I answered the “where do you see yourself in five years” question by saying “I hope to be a mad scientist.” Cheeky, but honest. Two years later – thanks to the support, resources, and room to explore provided by this program – I’m well on my way.

Thanks to Teemu Leinonen for finding a position for me to do this research in LeGroup, and for reading drafts. He and Anna Keune piled my desk with literature, giving me a jumpstart on getting perspective on my project within the context of people with similar goals through the years.

Thanks to Katian Witchger, Nuno Correia, and Tarmo Toikkanen for reading early versions of this manuscript and pointing me in interesting directions.

Jyri Pieniniemi is my Meemoo brain trust. He provided the original seed of the idea, and has bounced ideas with me for the past year. He also made the awesome illustration on the cover.

Thanks to Pascal Finette, Diane Bisgeier, and Mozilla WebFWD for taking on me and my project, showing me how the überwebgeeks throw down in Silicon Valley, and fighting for the open web.

My family and extended family around the John C. Campbell Folk School gave me a passion for making things and life-long learning.

Aino, here’s to a lifetime of tech and textile collaborations.

Tiniest but not least is Ilo, who came into the world a month ago. Thanks so much for the snuggle breaks. You inspire me.



CONTENTS

1	INTRODUCTION	1
2	CONTEXT	5
2.1	Hackers and Hackability	5
2.1.1	iOS	6
2.2	Metamedium	7
2.3	Tools	8
2.3.1	Direct manipulation	8
2.4	Programming for children (and other artists)	9
2.4.1	Logo (1967)	9
2.4.2	Smalltalk (1972)	10
2.4.3	Scratch (2006)	10
2.4.4	Processing (2001)	11
2.5	Visual programming languages	13
2.5.1	GRAIL (1969)	13
2.5.2	Audio/visual environments	13
2.5.3	On the web	14
2.6	Free Software Movement	15
2.7	Maker Movement	16
2.8	Plugin architecture	17
2.8.1	jQuery	17
2.8.2	Arduino	17
2.9	The Internet	18
2.9.1	Participatory media online	18
3	PREVIOUS WORK AND MOTIVATION	21
3.1	Early experiences	21
3.2	Voice Synthesizer (2002), Flash	22
3.3	Megacam (2010), Flash	23
3.4	Opera projection mapping (2011), Quartz Composer	23
3.5	Web Video Remixer (2011), HTML	25
4	DEVELOPMENT	27
4.1	Mozilla WebFWD	27
4.2	Software design for hackability	28
4.2.1	Common communication library for modules	28
4.2.2	Readable, sharable app source code	28
4.3	Usability design for hackability	29
4.3.1	Direct manipulation	29
4.3.2	What is abstracted	30
4.4	Graphic design	30
4.5	Persona and scenario-based design	30
4.6	Usability testing and feedback	32
5	APPS	35
5.1	Example apps	35
5.2	Economic system modeled with Meemoo	35
5.3	Live animation visuals for dance party	38
6	FUTURE DEVELOPMENT	41

6.1	Community for sharing apps	41
6.1.1	Socket communication	42
6.2	Touchscreen support	42
6.3	Code editing	42
6.4	Twenty Apps to Build With Meemoo	43
7	CONCLUSIONS	47
	Bibliography	49
	APPENDIX	53
A	THE PITCH	55
B	PERSONA PROFILES AND SCENARIOS	57
B.1	Cate the Creator	57
B.2	Henry the Hacker	57
B.3	Molly the Modder	58
C	CODE SAMPLES	59
C.1	Defining Inputs and Outputs (JavaScript)	59
C.2	Meemoo App Source Code (JSON)	60

INTRODUCTION

The power to understand, modify, and create new media tools should not be restricted to those with a freakish knack for manipulating abstract symbols.

— Paraphrased from Bret Victor’s “Kill Math” (2011)

The people that created the vision of the personal computer in the late 1960s wanted everybody to be able to create their own digital tools (see 2.2 on page 7). Traditional computer programming has a steep learning curve, so most people are satisfied with the tools that come with the computer or are offered as services online. The barrier to entry to learning traditional programming involves setting up a programming environment, learning different systems for creating presentation and logic, and learning different programming and markup languages and syntaxes.

People that overcome these barriers to entry are able to create software tools for themselves and others. These tools allow us to understand and modify our environment and culture. People that create tools that are open to modification afford other people the ability to learn from their tools and to build new tools on the old.

I wanted to combine my programming experience with visual programming and creative web applications to make a framework for my own web experiments. I decided to make something with creative value to a wider audience by considering audiences for the framework other than myself – programmers and non-programmers alike. Such a tool could invite people without coding experience to modify software tools and create their own. To this end, I designed a web-based visual programming framework called Meemoo.

The main objectives of this project:

- Design a modular dataflow visual programming framework using web technologies.
- The framework should afford non-coders the ability to modify creative web apps by configuring wires that represent how modules communicate.
- Apps created with the framework should have source code that is easy to read and share.
- Web coders should be able to extend the framework by creating modules using web standards. There should be a simple syntax to define the inputs and outputs of a module.

Within this framework, an *app* is a collection of modules and the wires that connect them. A *module* is a web page that can live anywhere online, and use any web technology. This web page includes JavaScript

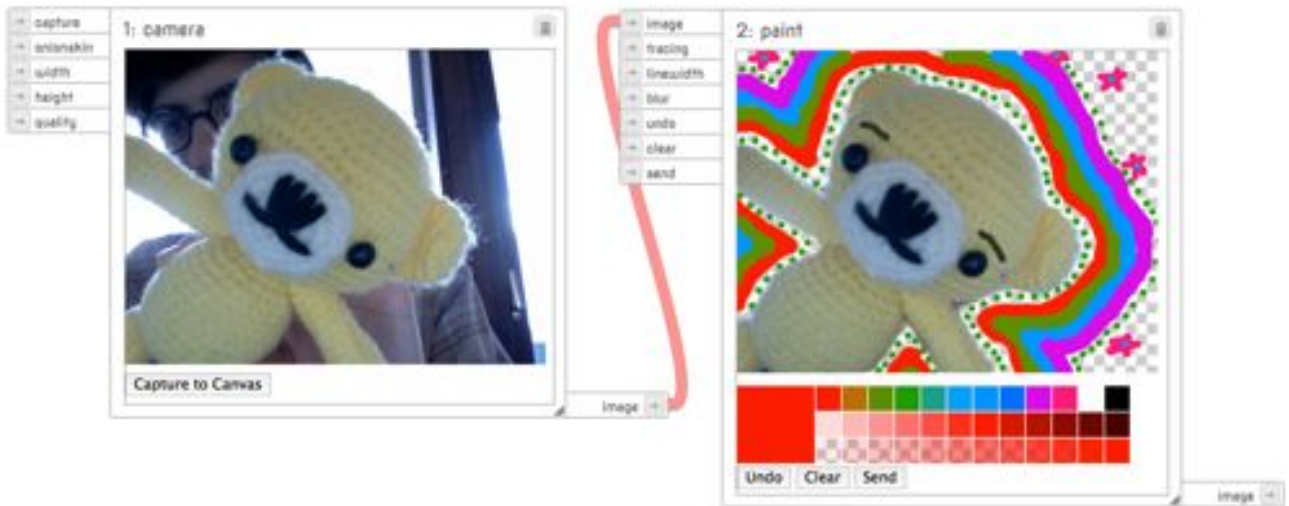


Figure 1: Meemoo screen-shot (March 2012)

that defines the module’s inputs and outputs: what data is accepted, how the data is processed, and what kind of data will be sent. The *wires* define where each module sends data. The source code of the app that defines an its layout, routing, and state can be saved and shared with a small amount of text.

Meemoo apps run in the web browser, so they will be able to run the same on PCs, smartphones, tablet computers, and soon TVs. Using, modifying, and making new apps are all done in the same browser interface, so no external developer tools are needed.

Meemoo apps are programmed by connecting modules with wires (Figure 1), putting programming within reach of non-coders.

In the course of this thesis I will refer to *programming* and *coding* as distinct skills. Learning to program is a process of learning to manipulate logical structures. Learning to code puts those structures into a linear-textual format that computers can parse. Different programmable systems emphasize and abstract these aspects differently. I will also use the term “people” to avoid the terms “user” and “developer,” as I would like to make this distinction moot in the context of the project.

So far I have focused module development on realtime animation tools, as this makes it simple to understand and engage creatively with the concept. It is not limited to animation though; any app or system that can be described by a dataflow graph can be made into a Meemoo app.

I did not start making Meemoo as an educational tool, but I have since been influenced by software projects inspired by Constructionist learning theory: Logo, Smalltalk, and Scratch. As I have read texts on Constructionism, I have seen parallels to my own experiences as a learner. I think that Meemoo has the potential to be a good tool for “learning by making.”

This thesis consists of a software project and a performance. The software project is Meemoo¹, which is a framework for hackable web apps. The performance consisted of live animation² made with the Meemoo framework (see 5.3 on page 38).

¹ Demos and source code available at <http://meemoo.org/>

² Live animation performance documentation: http://youtu.be/T_tCyYGLWKM

CONTEXT

Meemoo has many direct and indirect influences and precedents in the way that it has been designed.

2.1 HACKERS AND HACKABILITY

The Jargon File, a reference and glossary started in 1973, gives eight definitions for “hacker.”

hacker: n. [originally, someone who makes furniture with an axe]

1. A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary. RFC1392, the Internet Users’ Glossary, usefully amplifies this as: A person who delights in having an intimate understanding of the internal workings of a system, computers and computer networks in particular.

...

8. [deprecated] A malicious meddler who tries to discover sensitive information by poking around. Hence password hacker, network hacker. The correct term for this sense is cracker.

(Raymond, 2003)

The eighth definition, despite being deprecated in the Jargon File, has become the popular understanding of “hacker.” For the purpose of this thesis and project I will use and promote the first definition. In this context, “hackability” refers to design that encourages understanding of the workings of a system, in a addition to the ability to modify said system.

It might be a lost cause to try to reclaim this term from its common cultural understanding. The Maker Movement, which also places value in understanding and modifying systems and things, does not have such negative baggage with their moniker, as “make” and “maker” seem like more constructive terms. Hacking is related to tinkering and bricolage, but those terms do not seem as concrete. Although it is not perfect, I will stick to the term “hackability,” as I think that it encompasses the spirit that I want to promote with regards to software. There are other projects that are embracing this meaning as well, such as Hackity Hack!¹ and Mozilla Hackasaurus², both aimed at getting children to explore coding.

*Hackability: n.
design that
encourages
understanding and
modification.*

¹ Hackity Hack!: <http://hackety.com/>

² Mozilla Hackasaurus: <http://hackasaurus.org/>

Pekka Himanen describes “The Hacker Ethic” as a work ethic based on passionate curiosity. This drive to make and tinker is not limited to high tech systems; it manifests itself in other interests such as carpentry or textile craft. It stands in contrast to Max Weber’s “Protestant work ethic,” which is based on a sense of duty and responsibility. (Himanen, 2001, p. 3-8)

Designing for hackability implies respect. The designer of a hackable thing acknowledges that they cannot imagine every potential use, so they enable people to modify it to their will and connect it to other things. This quality can apply to software, physical artifacts, and services. For a physical artifact to be hackable, it should follow the Maker’s Bill of Rights (Figure 9 on page 16). For a service to be hackable, it should make its data freely available with an application programming interface.

For software to be hackable the source code should be available under a free license (see 2.6). While this enables other people with coding skills to modify a software project, I would like to expand the affordances of software hackability to non-coders.

2.1.1 iOS



Figure 2: App Icon

I saw the rise of touchscreen devices like the iPhone and iPad as a step backwards for participatory media. As originally marketed, these devices were designed primarily for media consumption (Jobs, 2007). The only media production capabilities afforded by these devices was taking and sharing photos. When Apple later opened up the App Store they took a timid step towards hackability by allowing third party developers to create apps that extend the functionality of the device. I say “timid” because only developers that pay for the privilege can write apps for these devices, and only apps that pass an opaque curation process are allowed in the App Store (Gruber, 2009).

Because of this closed ecosystem and technical limitations, the design of apps for iOS tend to have low to no hackability. In general, an app is designed to do one thing. The designer decides what the app does, how it communicates, where things can be shared. The “user” then uses the app. The demarcation is well-defined between the designer and user roles.

The standard icon for an app looks like a shiny glass object (Figure 2), which mirrors the aesthetics of the device itself. It symbolizes something highly designed and polished, not to be opened. For both the device and the apps that run on it, the design implies “no user serviceable parts inside.”

There are some apps designed for media creation. Garageband³ and Photoshop Touch⁴ for iOS are tools for music creation and image manipulation designed for touchscreens. These tools are each highly designed for one kind of media creation. These apps can only communicate with services that the designers have chosen.

³ Garageband: <http://www.apple.com/ipad/from-the-app-store/apps-by-apple/garageband.html>

⁴ Photoshop Touch: <http://www.adobe.com/products/photoshop-touch.html>

There are some apps that do allow some programming-based hackability: apps that are code editors. These include Codea⁵ by Two Lives Left for Lua, Processing.js Mini-IDE⁶ by Brian Jepson, and GLSL Studio⁷ by kode80 for OpenGL shaders. These three apps are text-based development environments that deal with the affordances and constraints of writing code on touchscreen devices in different ways. For example, Codea has touch-optimized number slider and color picker widgets embedded in the code to manipulate variables.

However, without an external keyboard, any kind of extended typing on a touchscreen device is a difficult task. It is also against Apple's regulations to load external scripts in native apps, which makes it hard to share code. Code that you write in these apps can only run in the "walled garden" of the app itself.

A different kind of programming might be better suited for touchscreen devices. Two potential alternatives to text-based programming are codeblocks (see 2.4.3 on page 10) and modular dataflow programming, like Meemoo.

Apple's iOS devices are powerful computers. It should be easier for people to hack and program their own tools.

2.2 METAMEDIUM

The Dynabook was a concept design developed at Xerox Palo Alto Research Center in 1968. The concept envisioned the personal computer more or less as we know it today. Alan Kay outlines some of the goals and philosophical influences of the project:

"Putting all this together, we want an apparently free environment in which exploration causes desired sequences to happen (Montessori); one that allows kinesthetic, iconic, and symbolic learning – 'doing with images makes symbols' (Piaget & Bruner); the user is never trapped in a mode (GRAIL); the magic is embedded in the familiar (Negroponte); and which acts as a magnifying mirror for the user's own intelligence (Coleridge)." (Kay, 1996, p. 33)

Alan Kay and Adele Goldberg coined the term "metamedium" to describe their vision of the computer as a medium that can be all other media. Unlike broadcast media which is passively consumed, computer media can also be participatory and active. This means that people can create and consume media with the same tool. (Kay and Goldberg, 1977, p. 393-394)

"I suggest that Kay and others aimed to create a particular kind of new media – rather than merely simulating the appearances of old ones. These new media use already

⁵ Codea: <http://twolivesleft.com/Codea/>

⁶ Processing.js Mini-IDE: <http://www.jepstone.net/blog/2010/04/16/processing-js-mini-ide-for-ipad-iphone-android-chrome/>

⁷ GLSL Studio: <http://glsstudio.com/>

existing representational formats as their building blocks, while adding many new previously nonexistent properties. At the same time, as envisioned by Kay, these media are expandable – that is, users themselves should be able to easily add new properties, as well as to invent new media.” (Manovich, 2008, p. 23)

It is common to consume and create media on the computer with tools made by other people. It should be easier to create, share, and modify our own tools.

2.3 TOOLS

“The ability to ‘read’ a medium means that you can access materials and tools generated by others. The ability to ‘write’ a medium means you can generate materials and tools for others. You must have both to be literate.” (Kay, 1990, p. 193)

Many digital tools have precedents in avant-garde art practices. In 1959 Brion Gysin manually “hacked” text with scissors, recombining the fragments to see what might be found (Burroughs, 1961). Word processing software can be seen as a tool that affords greater hackability to text. In 1972 DJ Kool Herc manually mixed two records to give partygoers a longer percussive break for dancing (Hermes, 2006). Non-linear audio editors afford greater hackability to music, which has led to different kinds of remixing. Although the art practices did not directly lead to the software tool development, both come from the same impulse to combine and recontextualize the world around us. (Manovich, 2003, p. 13)

These digital tools are designed to afford hackability to one particular kind of media. Meemoo is designed to afford hackability to the tools themselves. Meemoo is a toolmaker that makes it easier for people to create and modify their own digital tools, and therefore to invent new media.

2.3.1 *Direct manipulation*

Bret Victor gave a talk called “Inventing on Principle” at the 2012 Canadian University Software Engineering Conference in which he demonstrated an experimental code editor (Victor, 2012). His editor shows the output of the code in real-time, without needing to change windows, compile, or save and refresh. Any number or color in the source code opens a slider or color picker, which immediately changes the visual output (Figure 3). As somebody with experience coding visual applications in JavaScript, Flash, and Processing, this demonstration was like seeing the light. Suddenly I could see that each mode change in my normal workflow is a waste of time and mental energy – the need to save or recompile, change modes, and refresh between changing code and seeing the result. People that make

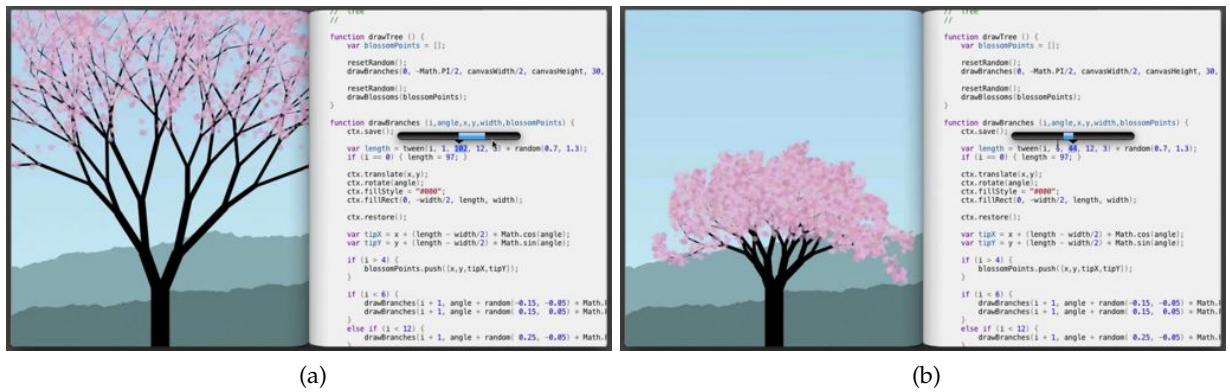


Figure 3: Bret Victor's direct manipulation of code

code editors should consider including this kind of direct manipulation.

2.4 PROGRAMMING FOR CHILDREN (AND OTHER ARTISTS)

While Meemoo was not designed specifically for children, it shares with these projects the goal of lowering the barrier to entry to programming.

2.4.1 Logo (1967)

Seymour Papert studied under Jean Piaget, an educational philosopher who outlined stages of mental development into a model of learning called "constructivism." The basic idea is that people construct knowledge structures through experiences, as opposed to being taught the structures. Papert added to this model, proposing that learning happens best when "the learner is consciously engaged in constructing a public entity, whether it's a sand castle on the beach or a theory of the universe" (Papert and Harel, 1991, p. 1). This idea is called "constructionism."

Papert realized that the computer, as metamedium, could be a powerful learning tool if students were able to create their own programs. Logo was designed as a simplified programming language for exploring mathematics. The first tests of Logo in the classroom predated the personal computer, sending code from a teletype terminal in the classroom to a remote mainframe computer (Papert and Harel, 1991, p. 6).

As computers became smaller and more common in classrooms, the signature Logo turtle was added to the system. This was a graphical representation of a turtle that would draw lines on the screen based on the instructions given by the child. For example, "repeat 5 [fd 100 rt 144]" tells the turtle "do this five times: walk forward 100 units, then turn 144 degrees to the right." This small program draws a star (Figure 4). By making the commands relative to the current position of the turtle, the language is easier to learn than

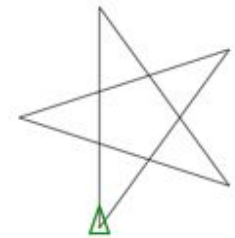


Figure 4: Logo turtle drawing a star

a graphical drawing system based on Cartesian coordinates. (Papert, 1993, p. 171)

2.4.2 *Smalltalk (1972)*

A fundamental requirement for the DynaBook research project was to create a system that could be programmed by the user. Kay invented Smalltalk and object-oriented programming for the Dynabook prototype system to lower the barrier to entry for coding. He was inspired by seeing children program with Logo, but wanted them to be able to program more complex tools. “Procedural turtle graphics just wasn’t it.” (Kay, 1996, p. 26) Object-oriented programming splits code into logical classes that define the data for the object and the methods that access or modify that data. By splitting the code like this it becomes easier to organize and navigate more complex projects.

In tests some children programmed their own tools, like a twelve-year-old girl’s painting application and a fifteen-year-old boy’s circuit design application. Kay later referred to these impressive results as “early success syndrome.” “The successes were real, but they weren’t as general as we thought.” Kay later decided that learning to program might be as difficult as learning to write, and take years to build up the mental models necessary to do it correctly. (Kay, 1996, p. 26)

2.4.3 *Scratch (2006)*

Text-based programming languages have requirements for syntax and indentation that if not followed perfectly will result in programs that do not run as expected (or at all). This can be frustrating for beginners and experienced coders alike. Most of my programming errors are missing semicolons or mismatched brackets. The creators of Scratch designed a coding system that works around syntax and the frustration of syntax errors.

Scratch uses drag-and-drop “code blocks” instead of a text-based syntax, which makes coding less error-prone for beginners. These code blocks snap together only in ways that make syntactic sense (Figure 5a). “Control structures (like `forever` and `repeat`) are C-shaped to suggest that blocks should be placed inside them. Blocks that output values are shaped according to the types of values they return: ovals for numbers and hexagons for Booleans. Conditional blocks (like `if` and `repeat-until`) have hexagon-shaped voids, indicating a Boolean is required.” (Resnick et al., 2009, p. 63)

Although creating a script with code blocks is more like snapping Legos together than writing code, it is still coding. The shape of the control structures is a direct metaphor to how code syntax works, and more visually obvious than brackets or indentation. I imagine these logical structures are transferable to textual coding.

I feel a little cheated to not have had Scratch when I was a child. I would have loved it.

I tend to program something with the Fibonacci sequence with every new language that I try. To test Scratch I made an absurdist animation of a cat running into walls and reciting Fibonacci numbers⁸ (Figure 5). I did not have this final output planned from the start. The available blocks influenced the direction of my exploration. For example, the last change was adding the drum sound when I saw that it was as easy as adding one more block to the script. Making something with comparable collision detection, color cycling, and audio triggering in Flash or Processing would have taken much longer.

This was my first Scratch project, and from launching the environment for the first time it only took about thirty minutes to snap it together. Granted, I am an experienced coder, but I try new languages and coding systems on occasion, and Scratch was by far the fastest and easiest to create something interesting. I look forward to playing with it more.

In Scratch each element on the screen is a “sprite” with its own variables and scripts. This makes Scratch compositions object-oriented by default, as it would be hard to do it any other way.

Code can be changed as the project is running, and the results change immediately⁹. This brings direct manipulation to coding, in that changes in the code can be seen in real-time, without recompiling or refreshing. This is an uncommon feature in programming environments. I imagine it is helpful for beginning programmers.

2.4.4 *Processing (2001)*

Processing was designed to teach programming fundamentals with a focus on visuals, motion, and interaction. The creators of Processing wanted it to extend the computer as a “medium for expression.” (Reas and Fry, 2007, p. 2-3) Processing was ported to JavaScript in 2008 (Resig, 2008).

“Software holds a unique position among artistic media because of its ability to produce dynamic forms, process gestures, define behavior, simulate natural systems, and integrate other media including sound, image, and text.”
(Reas and Fry, 2007, p. 1)

Processing is a textual programming language and environment based on Java. After using Scratch, I am surprised that Processing and Flash were the first languages introduced to students at Media Lab Helsinki. I think that Scratch is a better first introduction to coding, because it abstracts the syntax away and lets the beginner focus on learning the logical structures of programming. I would recommend anybody teaching programming to beginners to start with Scratch, at least for the first lesson.

⁸ Fibonacci Cat! by the author: <http://scratch.mit.edu/projects/forresto/2398409>

⁹ This works best with some kinds of code edits, like changing variables or colors. Moving larger blocks sometimes requires restarting the project.

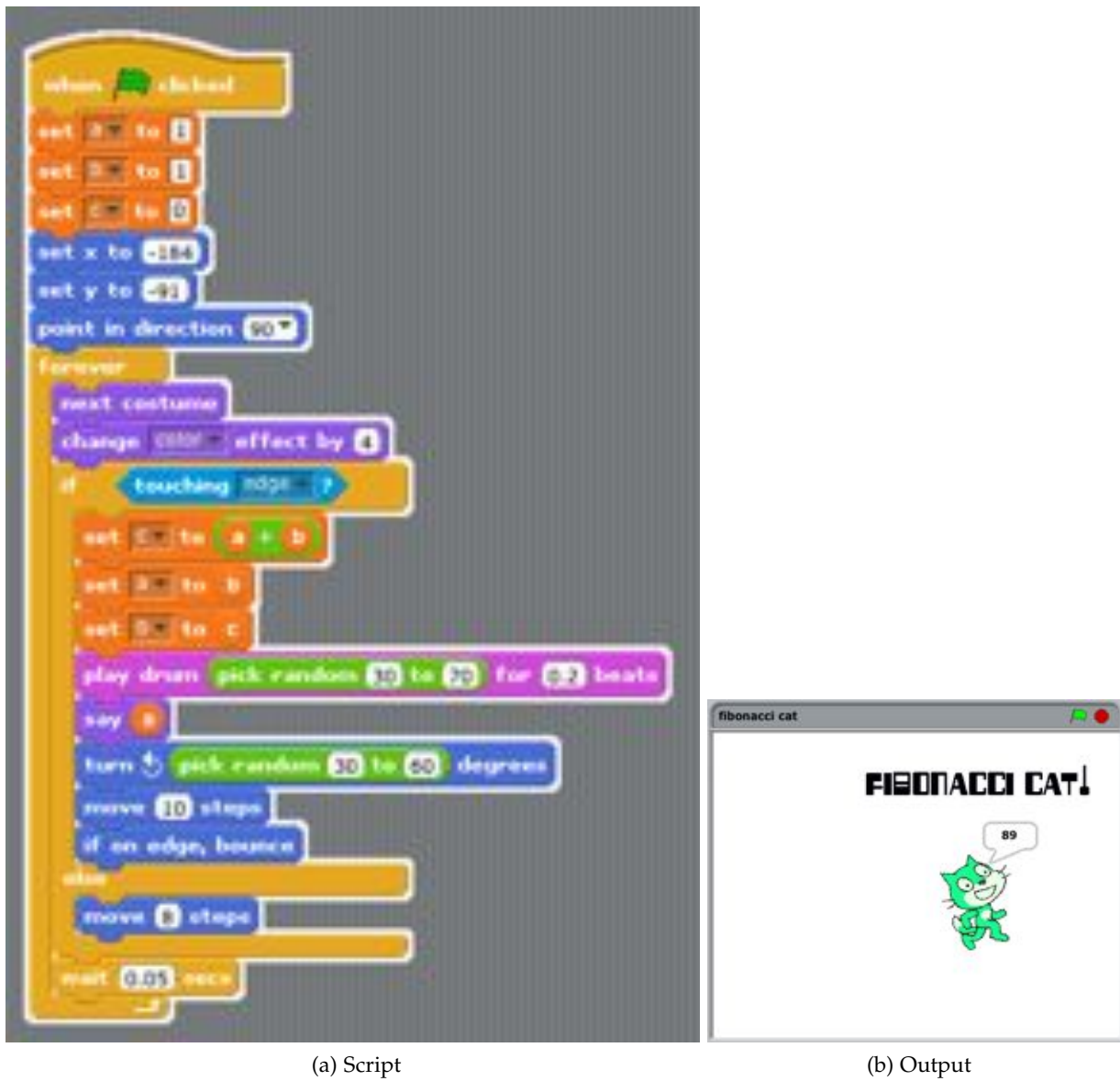


Figure 5: Fibonacci Cat! Scratch program by the author

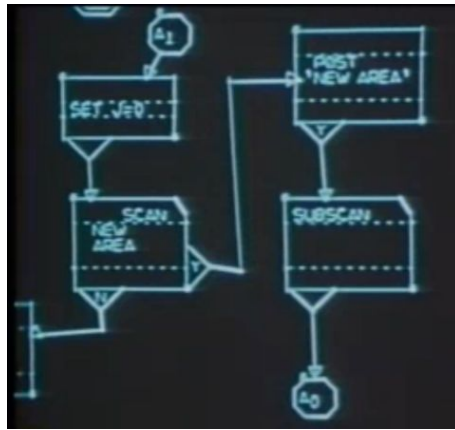


Figure 6: GRAIL

2.5 VISUAL PROGRAMMING LANGUAGES

Meemoo is a kind of dataflow visual programming environment. This means that programming is done by connecting modules with wires. The modules can hold and process arbitrary data: text, audio, images. The wires define and visualize how data moves from module to module.

2.5.1 *GRAIL (1969)*

GRAIL (GRAphical Input Language) was an early experimental dataflow environment developed by the Rand Corporation from 1967 to 1969 (Figure 6). This interface was driven by a graphics tablet, so everything could be done without a keyboard. Nodes were added by drawing a box in place. Edges were drawn from node to node. Labels were added to the nodes with handwriting recognition. Edges were disconnected by scribbling over them. (Ellis et al., 1969, p. 3)

Kay credited the project with directly inspiring some of the user interface elements in the DynaBook system, like windows that were resizable by dragging the corner. This feature was later adopted by Macintosh and later Windows operating systems. “It was direct manipulation, it was analogical, it was modeless, it was beautiful.” (Kay, 1996, p. 10)

2.5.2 *Audio/visual environments*

Visual programming is used most in the domain of real-time audio processing and synthesis (Pure Data and Max/MSP), generative animation and visual effects (Quartz Composer and vvvv), and 3D material and shading design (Softimage Interactive Creative Environment) (Morrison, 2010).

This is probably due to the fact that people involved in audio/visual production tend to be comfortable with connecting equipment with cables, so it is easier to learn a system based on this metaphor

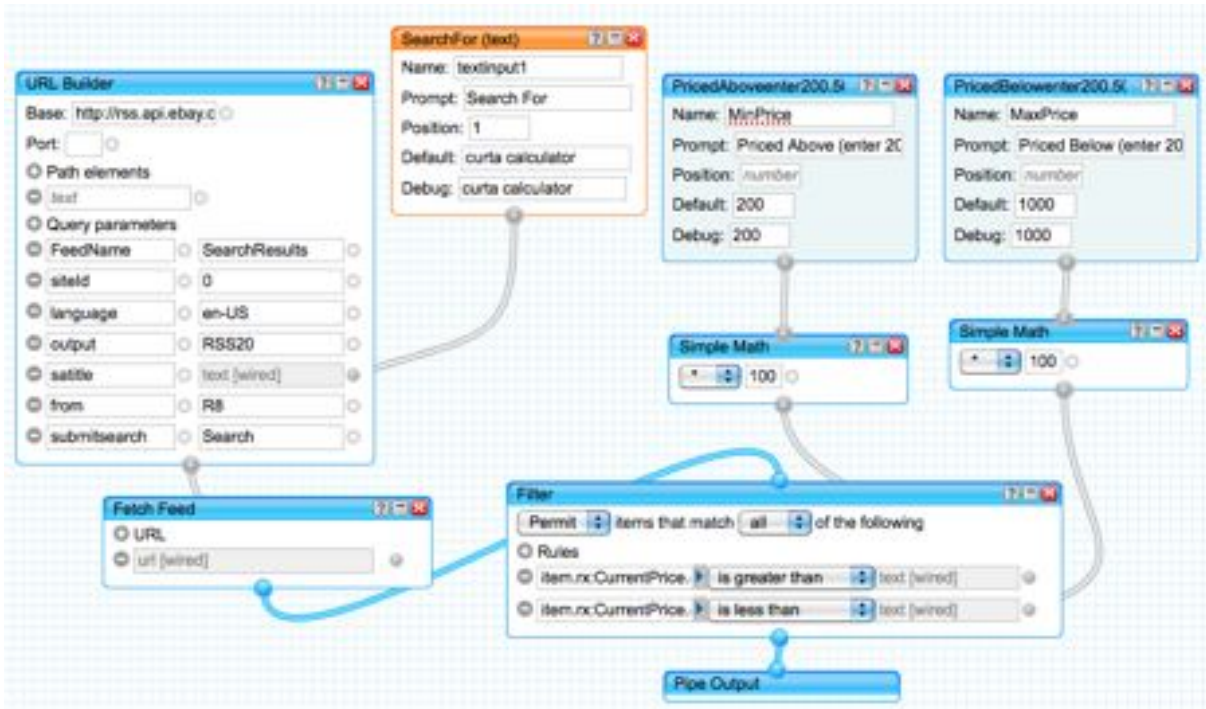


Figure 7: Yahoo! Pipes layout that watches eBay for items with a search term and price range

than one based on linear-textual coding. Also, these systems tend to provide immediate feedback when modifying their layout.

2.5.3 On the web

One precedent for web-based visual programming is Yahoo! Pipes (Figure 7). This web application was released in 2007 with the goal of enabling people to “assemble personalized information sources out of existing Web services and data feeds” (Sadri et al., 2007). Pipes is designed for asynchronous data processing, which makes it good for tasks such as setting up alerts based on searching a service.

In contrast to this system for asynchronous data processing, Meemoo is designed to make real-time interactive audio/visual applications. The open modular design of Meemoo allows its capabilities to keep pace with the capabilities of web browsers.

ThreeNodes.js (Figure 8) is a visual programming environment for web-standards-based 3D graphics. It was released in September of 2011. The focus is on building, shading, and lighting a 3D environment. Once you program the scene visually you can export the resulting JavaScript code.

ThreeNodes.js is much more specific in its design and output than Meemoo. In the future I will be working with this project to make some of its modules work within Meemoo.

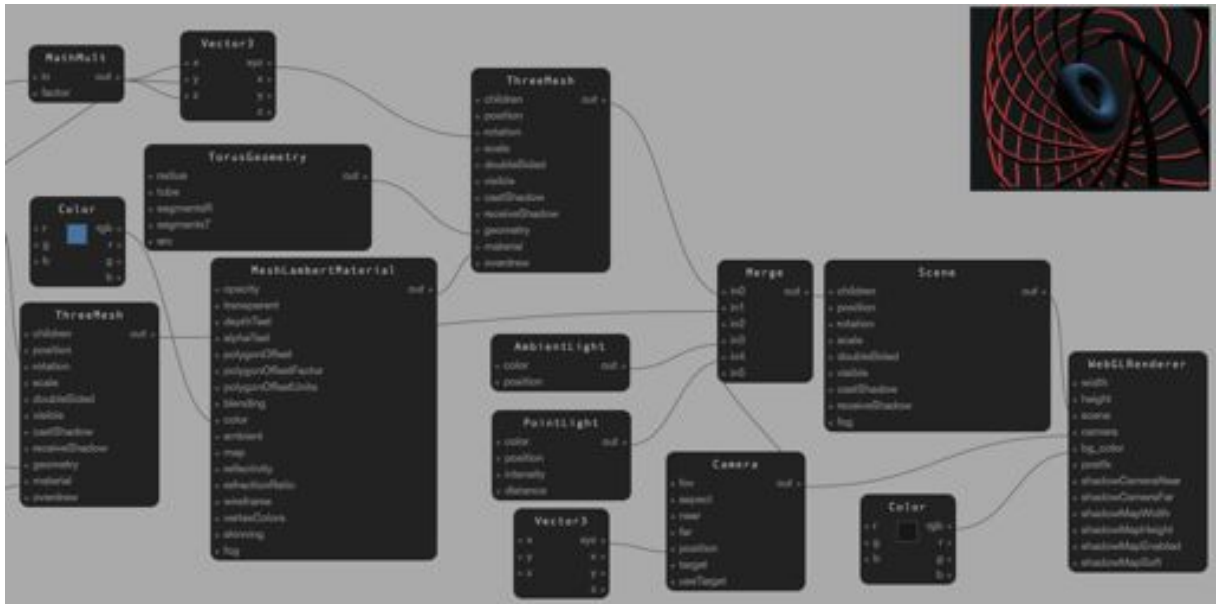


Figure 8: ThreeNodes.js

2.6 FREE SOFTWARE MOVEMENT

Richard Stallman wrote the “The GNU Manifesto” in 1985 at the start of the GNU project. In it he explains that software should be free, meaning that the source should be available to examine and modify. He framed his argument in terms of personal honor, “I consider that the Golden Rule requires that if I like a program I must share it with other people who like it.” He also advanced the practical argument that free software would prevent “wasteful duplication” of programming effort which could instead be used “advancing the state of the art.” (Stallman, 1985)

In my programming experience, this has proven to be the main benefit of free software. Meemoo relies heavily on two JavaScript libraries that are free software, jQuery¹⁰ and Backbone.js¹¹. Both of these libraries are written by communities of people who are collectively much better at JavaScript than me. They design these libraries to be helpful for other programmers, and give them away under open-source licenses.

jQuery seeks to abstract away browser differences, making it much easier to write code that runs the same on different browsers. Backbone.js provides structure for the data models, views, and collections that are required to make a maintainable object-oriented web application. This makes it easier to keep track of the data structures and state of the apps, modules, and wires in Meemoo.

I can confidently say that without these libraries Meemoo would not exist. My programming knowledge is built on viewing and using source code made by other people. Meemoo is built on libraries made by other people. This cumulative innovation is referred to as

¹⁰ jQuery: <http://jquery.com/>

¹¹ Backbone.js: <http://documentcloud.github.com/backbone/>



makezine.com

THE MAKER'S BILL OF RIGHTS

- Meaningful and specific parts lists shall be included.
- Cases shall be easy to open. ■ Batteries shall be replaceable. ■ Special tools are allowed only for darn good reasons. ■ Profiting by selling expensive special tools is wrong, and not making special tools available is even worse. ■ Torx is OK; tamperproof is rarely OK.
- Components, not entire subassemblies, shall be replaceable. ■ Consumables, like fuses and filters, shall be easy to access. ■ Circuit boards shall be commented.
- Power from USB is good; power from proprietary power adapters is bad. ■ Standard connectors shall have pinouts defined. ■ If it snaps shut, it shall snap open. ■ Screws better than glues. ■ Docs and drivers shall have permalinks and shall reside for all perpetuity at archive.org. ■ Ease of repair shall be a design ideal, not an afterthought. ■ Metric or standard, not both.
- Schematics shall be included.

Make:
technology on your time

Drafted by Mister Jalopy, with assistance from Philip Torrone and Simon Hill.

Figure 9: The Maker's Bill of Rights

“not reinventing the wheel” and “standing on the shoulders of giants” (Williams, 2012, p. 138).

Free and open-source libraries are the foundation of this project, but it does not stop there. Free software powers the majority of the web. As of March 2012 65% of web servers run the free Apache HTTP Server, versus 14% that run Microsoft's closed-source server software (Netcraft, 2012). There are several open-source languages (PHP, Ruby, Python) that run on those servers and many open-source frameworks (Symfony, Rails, Django) written in those languages. There are two competitive free browser implementations that push web standards and innovation: Gecko (Firefox) and Webkit (Safari and Chrome). Firefox is “the most widely used consumer-facing piece of free software” and Webkit has the backing of two of the largest technology companies, Apple and Google (Villa, 2010).

2.7 MAKER MOVEMENT

The Open Hardware and Maker movements value hackability in electronics and other physical objects. The Maker's Bill of Rights (Figure 9) from Make Magazine lists several “commandments” for makers and hardware manufactures to consider. (Mister Jalopy et al., 2005)

Although Meemoo is a software project, it is designed to follow the spirit of this document. The “case” of the app is easy to open and see the wiring view. The wiring view is analogous to an electronics schematic, and is always “included” by design of the framework. I am planning on using a screw icon (Figure 10) for components that you can “open” and view the source.

2.8 PLUGIN ARCHITECTURE

Successful frameworks are designed to enable community members to make and share components that extend the capabilities of the framework. Firefox has add-ons, Facebook has apps, jQuery has plugins, Processing has libraries, and Meemoo has modules.

2.8.1 *jQuery*

jQuery is hackable by virtue of its open-source license, but it is also hackable because it provides mechanisms for programmers to extend it with plugins. There is no official repository of plugins, but one site dedicated to jQuery¹² claims to have over 2000 listed in their directory. The only JQuery plugin that I am using for Meemoo is the official JQueryUI¹³, which makes it simpler to implement drag-and-drop functionality and styled buttons.

The way the Meemoo framework accepts modules is analogous to JQuery’s plugin architecture. Anybody can extend the framework by making a module. Meemoo provides the framework and a communication standard for the modules.

2.8.2 *Arduino*

Arduino was designed in 2005 to make it easier to work with electronics. Programming can be done from a Mac or PC via a standard USB cable, where earlier microcontrollers would require expensive programming hardware. It was designed to be affordable to start: \$30 compared to \$100 for earlier microcontrollers (that were less powerful). They released the schematics under a Creative Commons license. Massimo Banzi, one of the creators, says that the most important impact of the Arduino is “the democratization of engineering.” (Kushner, 2011)

A cottage industry has sprung up around Arduino. In much the same way that any JavaScript developer can write a plugin for JQuery, anybody with knowledge of electronics can make modular “plugins” for Arduino. Two examples are Gameduino¹⁴ and Pulse Sensor¹⁵. Both of these projects were funded on Kickstarter and raised many



Figure 10: Open me!

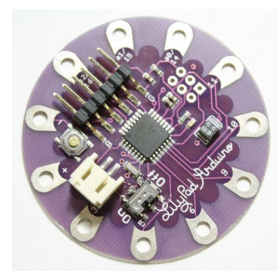


Figure 11: Lilypad Arduino

¹² jQuery4u plugins directory: <http://www.jquery4u.com/plugins/>

¹³ JQueryUI: <http://jqueryui.com/>

¹⁴ Gameduino raised 11.5x their goal: <http://kck.st/f44kHG>

¹⁵ Pulse Sensor raised 6.1x their goal: <http://kck.st/nae9lR>

times their funding goal, which shows the demand and interest in the Open Hardware community. Gameduino is a “shield” that plugs into Arduino and contains the needed electronics and software libraries to create vintage-style video games. Pulse Sensor is a simpler module that contains the electronics needed to measure heart rate from a fingertip or ear lobe. Plugging both of these modules into an Arduino would take a hacker a long way towards creating a pulse-controlled video game.

Arduino is a framework that simplifies electronic engineering. The Arduino community makes modules that can interoperate through that framework. Similarly, Meemoo is a framework that simplifies web app development. People will be able to experiment with creating apps by “plugging” modules together in the framework, and they will also be able to invent and share their own modules.

2.9 THE INTERNET

People learn by imitating and communicating with people with more skill. Lev Vygotsky refers to the difference between what one can do independently and what one can do with guidance the “zone of proximal development” (ZPD). This theoretical space is where learning happens, and can be seen in different forms in work, play, and in the classroom. (Vygotsky, 1933)

The Internet provides a social communication framework where anybody connected can learn from others: the ultimate zone of proximal development. Like most of my programming projects, Meemoo has pushed me into the ZPD with several challenges, stretching my capabilities as a programmer. Stack Overflow¹⁶ has been an especially helpful community for learning from the questions and answers of others.

2.9.1 *Participatory media online*

The term “Web 2.0” was defined and clarified by Tim O’Reilly in 2005. He outlined several characteristics of the companies and services that survived or were successful after the dot-com bubble crash of 2001. These include providing web apps as services, controlling user-generated data, and harnessing collective intelligence. (O’Reilly, 2005)

One aspect of this trend is the rise of online services that facilitate publishing content. This began with blogging services like LiveJournal in 1999, photo sharing sites like Flickr in 2004, and video sharing sites like Youtube in 2005. These services helped make the web more participatory, giving any person with internet access the ability to publish text, images, and video. These participatory platforms make media distribution easier by abstracting away the need to learn about web servers and HTML.

¹⁶ The author’s profile on Stack Overflow: <http://stackoverflow.com/users/592125/forresto>

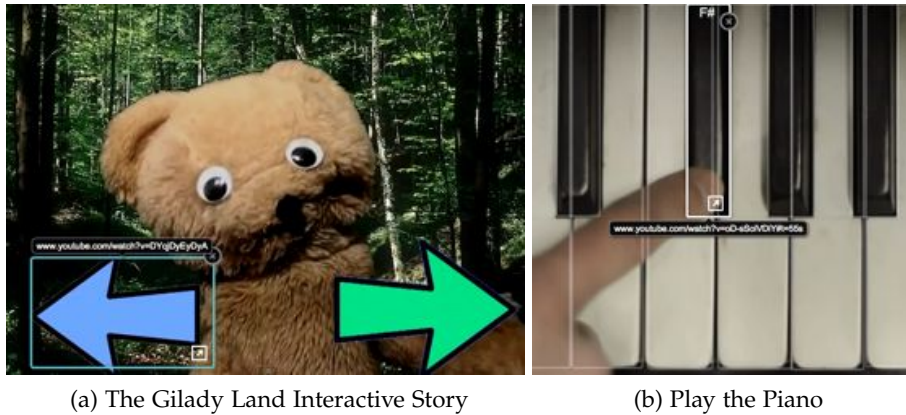


Figure 12: Interactive video with hyperlinking

While these services enable publishing of content, they are limited in how they can be used. The typical service presents a form with input fields for title, media file, description, and tags. This information then creates a single web page.

Some people have worked within these constraints to create interactive media using hyperlinks. For example, Youtube user TimsPuppetPals made a collection of videos called “The Gilady Land Interactive Story.”¹⁷ One video is the entry into the story, and the rest of the videos are unlisted within the Youtube system. In the end of each section of the story, viewers are presented with two choices as hyperlinks within the video (Figure 12a). Interactive stories based on hyperlinking are limited to this kind of choose-your-own-adventure branching storyline.

Another example of simple interactivity with hyperlinks is “Play the piano”¹⁸ from Youtube user kokokaka3000. This interactive video uses hyperlinks overlaid on each key of piano keyboard. As you click on the links above each key, the video skips to a finger playing that key (Figure 12b).

In order to create interactivity more complex than these two examples, some form of programming is needed, and it must be hosted outside of the service.

An example of a project with more complex interactivity is Darren Solomon’s “In B Flat.”¹⁹ To create this project, Solomon solicited videos of people making simple music in the same key. He then embedded twenty of these videos in a grid in a web page (Figure 13). To interact with the piece, you press play on any or all of the videos in any order. Because of the floating nature of the music in all of the samples, they tend to sound good together no matter how they are mixed.

The structure of “In B Flat” – multiple videos that can be independently controlled in one HTML page – can be considered a new media afforded by the participatory nature of Youtube, and the ease

¹⁷ “The Gilady Land Interactive Story” <http://youtu.be/spVMYoUcuR4>

¹⁸ “Play the piano” <http://youtu.be/oD-sSolVDiY>

¹⁹ “In B Flat” <http://www.inbflat.net/>

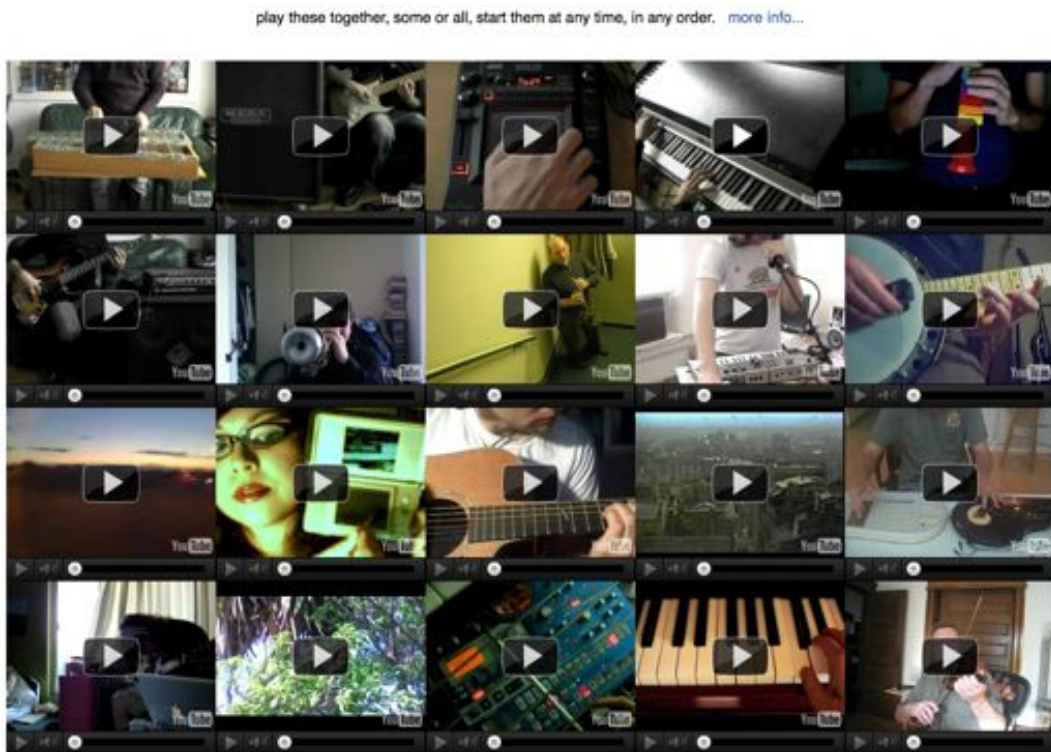


Figure 13: In B Flat

of embedding videos. The ability to create this new media required HTML coding knowledge.

Youtube's embeddable player has a JavaScript Player API (Application Programming Interface)²⁰ which makes more complex interactivity possible. Flickr's API²¹ has enabled developers to create applications that can access and edit Flickr's massive database of photos and metadata²². These APIs increase the web's hackability, but only for people with coding skills.

Meemoo is a web app that uses and extends media from other Web 2.0 services, without textual coding. For example, animated GIFs created with Meemoo can be saved directly to Imgur, an image-sharing site. Youtube videos like "Play the piano" can be used to make a musical composition with a step sequencer. Source code for Meemoo apps will be easy to share anywhere that text can be posted.

²⁰ YouTube JavaScript Player API https://developers.google.com/youtube/js_api_reference

²¹ Flickr's API: <http://www.flickr.com/services/api/>

²² An example Flickr application by the author: <http://taggraph.com/>

PREVIOUS WORK AND MOTIVATION

My motivation to make this project comes from years of experimenting with digital technologies. I have worked in different languages and environments, but the ability to share my work online has always brought me back to working with web technologies. I make my experiments into online creative tools (web apps) in order to see how other people use my creations.

In a way, Meemoo is an abstraction of all of my earlier digital creative experiments. I plan on rebuilding some of these experiments in Meemoo to make it easier for me (and others) to modify how they work.

3.1 EARLY EXPERIENCES

I will start with an abridged history of my relationship with digital media. Three anecdotes will illustrate the three most important aspects of Meemoo: creative, hackable, web.

My first memory of interacting with a computer was with an Apple Macintosh that my father brought home from work in the mid-1980s. I have a strong visual memory of using the mouse to connect numbered dots to draw a star. Once the star was complete it briefly became animated. Seeing this graphic, however simple, react to my input and then come alive captured my imagination. We only had that computer for a few days, but I was hooked.

In the course of researching this paper, I was able to trace this distant memory to its source, and find a screenshot (Figure 14). This interaction was part of an introductory program to teach mouse skills to novices, called “Mousing Around.” It ran automatically on the first boot of new Macintosh computers. (Wichary, 2005)

Because of timing or school priorities, I was not part of the small generation of students that was exposed to BASIC or LOGO programming in school. I remained interested in computers, spending any time that I could get my hands on them on shareware games and paint programs. I did not get into programming until high school, in two very different ways: Texas Instruments graphing calculators and web programming.

My higher-level math classes used TI-8x series of graphing calculators. These have the ability to write and run programs with a BASIC-like syntax. My first program mirrored a game played by many children on standard calculators: the “+1 game.” This game is played by pressing the buttons [1] [+] [1] [=], and then pressing [=] as fast as

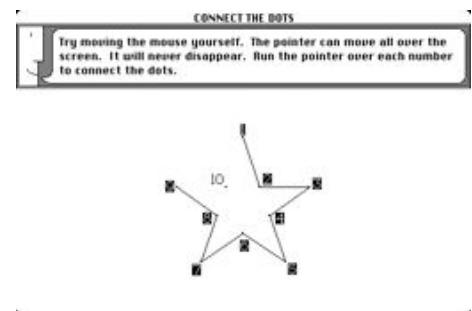


Figure 14: Mousing Around

possible. This makes the calculator into a counter, and we would have races to see who could press [=] fastest. Pressing buttons seems to be a common interest for children. When a system reacts to the button press, it gives the child a sense of control. The program that I wrote was just a few lines of code. It counted from zero, adding one and displaying the result in an infinite loop as fast as the calculator could go. I had automated the +1 game, taking out the button-pressing dynamic. It was satisfying to see the numbers flying by on the screen. I then made a new version of the script that printed the Fibonacci sequence in the same manner.

I then figured out how to script complicated graphic drawings on the small monochromatic screen. I would watch with interest as the calculator slowly rendered patterns from my scripts, one stroke at a time. This was my first experience with programming graphics. I never managed to make a program draw what I originally had in mind, but this was not discouraging. The serendipitous images that emerged from my experiments encouraged me to explore different directions, and create new challenges for myself. I learned about cartesian geometry, algebra, and logic from these code explorations.

The scripting environment on these calculators gave them hackability. This afforded my creative exploration of math and graphical programming in ways that Texas Instruments had not specifically designed.

The availability of the Internet in my home spurred the second programming interest. It was empowering to publish my first web site. It was a place where I could freely express myself in many different ways. Anybody in the world could see it, through the same window and at the same resolution as the websites of corporations, governments, and universities. Learning how to create and post webpages gave me a level of active participation that other media had not offered.

I learned web programming by example, mostly thanks to the “view source” command on the browser. I would take a little bit of code from a tutorial, some code from another page’s source, and tinker and experiment with the combination in an editor that showed both the code and output in the same window. These web programming experiments continued from this time and have culminated in this thesis project.

3.2 VOICE SYNTHESIZER (2002), FLASH

This project was the most ambitious programming that I had done at the time. The result was hacked-together voice synthesizer¹ that used looped samples of me saying all of the basic English phonemes. As you type on the keyboard, the audio for each phoneme plays and the image changes to my mouth making that sound (Figure 15).

¹ Voice Synthesizer: <http://forresto.com/oldsite/interactive/mbx/>

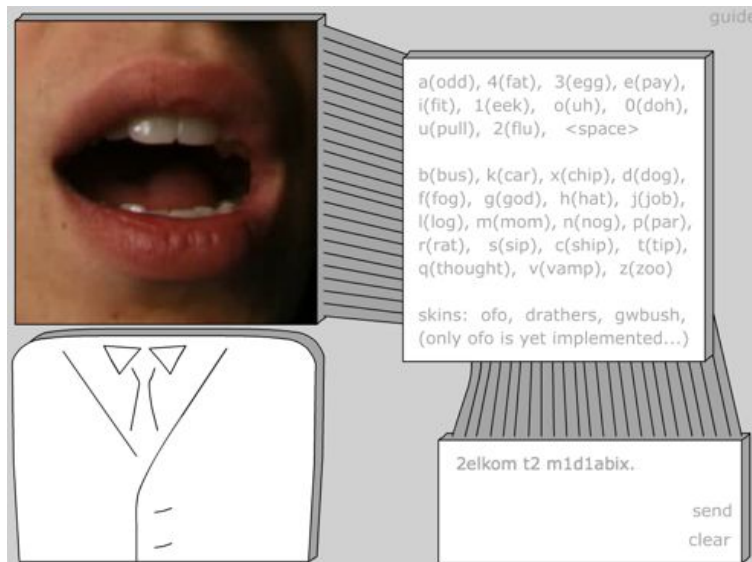


Figure 15: Voice Synthesizer

I originally imagined making different voices and faces of newscasters and politicians as plug-ins to the system. Because of lack of time and programming skill, this never happened. It would be easy to make such a system in Meemoo, with the plug-ins as modules containing web video and a listing of timecode triggers for each phoneme.

3.3 MEGACAM (2010), FLASH

Inspired in part by Lomo cameras, I made a series of webcam toys in Flash². One toy makes an arbitrary number of exposures in a grid with different patterns (Figure 16). Another is a kaleidoscope with changeable mirror size. Another takes slit-scan photographs.

In the early prototypes I had sliders for all of the variables. I chose four presets for the variables to make the interface simpler, but that also removed the possibility for me and other people to experiment with the variables. I designed away the hackability.

If Megacam were remade as a Meemoo app, it would be possible to keep the simplicity of the presets, but also the hackability of exposing all variables. People could also wire in image effects to change the kinds of photos that the app makes.

3.4 OPERA PROJECTION MAPPING (2011), QUARTZ COMPOSER

Last year I was working on a multi-screen video projection system for the set design of an Opera³. I found Quartz Composer modules

² Megacam: <http://sembiki.com/megacam/>

³ Set design: <http://crucible.mlog.taik.fi/productions/unperformed-a-performance/>

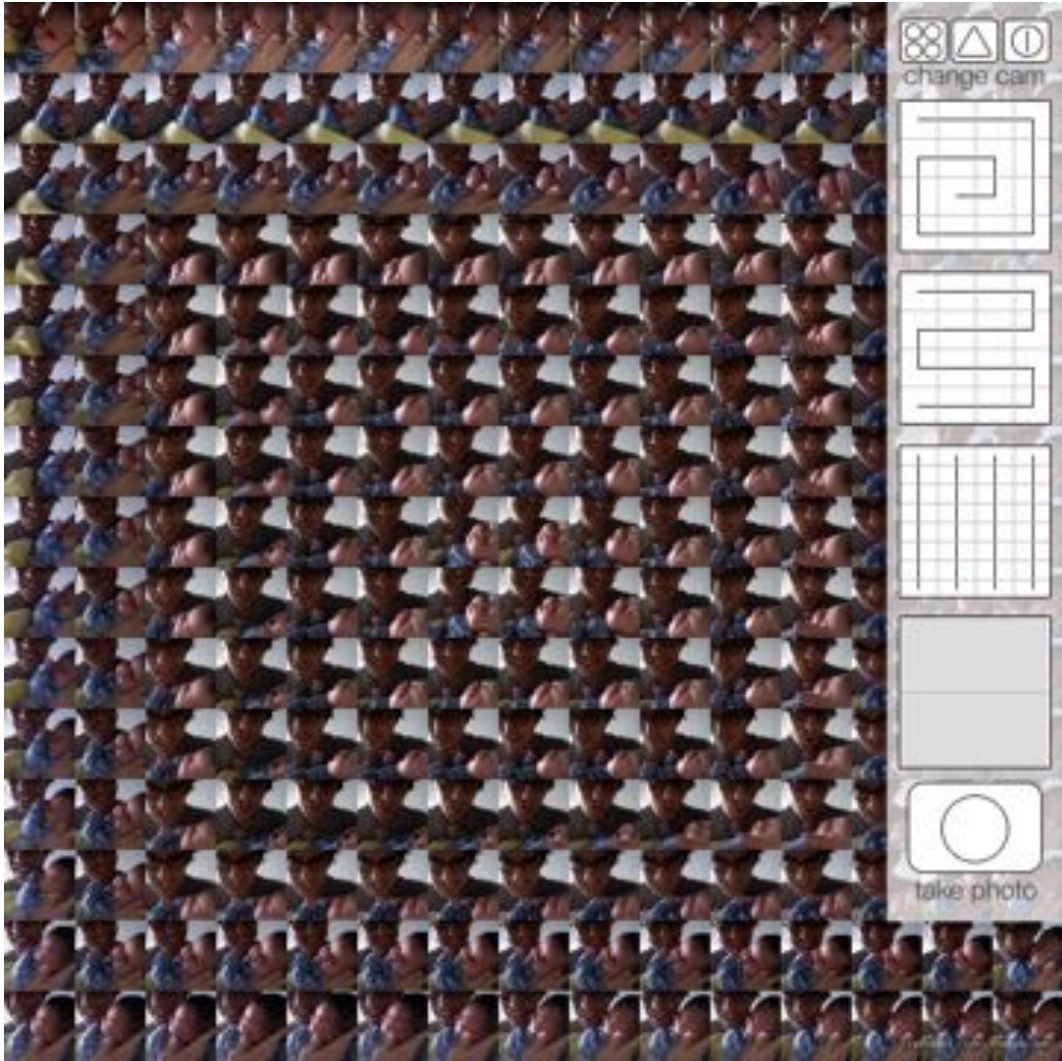


Figure 16: Megacam

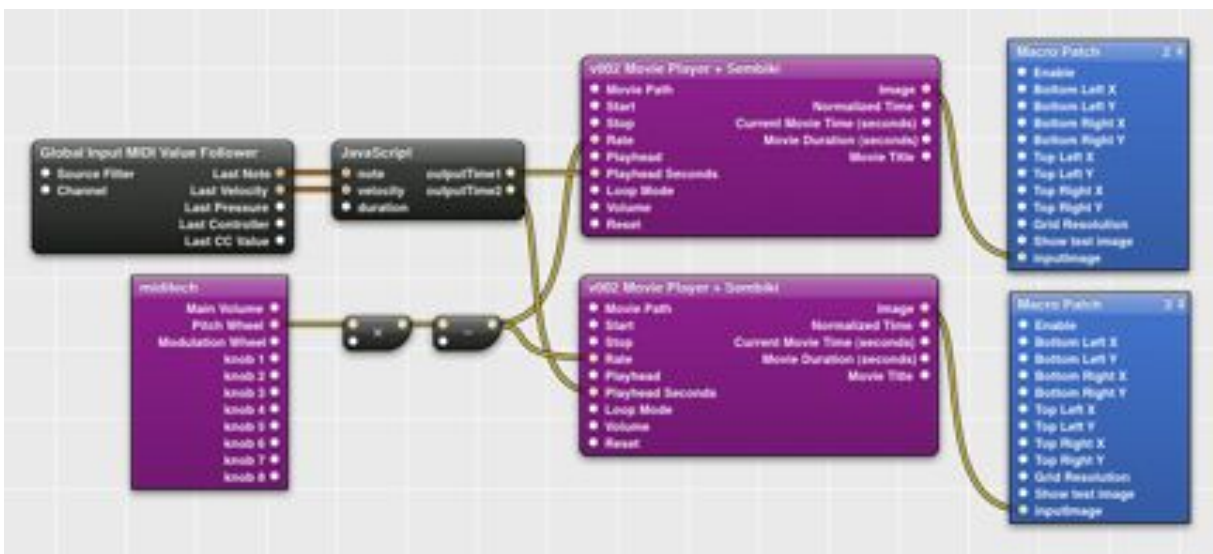


Figure 17: Connecting open-source modules in Quartz Composer

for midi control, video playback⁴, and projection mapping⁵. I patched them together (Figure 17) to create a system that controlled video on four projection-mapped screens from one projector.

These modules were all shared online by their authors in the open-source spirit. I needed to add a feature to the video player module, and was able to do so in XCode, Apple's proprietary development environment.

Developing this application required searching the web for the modules, modifying and recompiling one of the modules in a development environment, and finally wiring the modules together in another environment. I was able to make this system with much less effort than if I had needed to write all of the software myself. However, it was several layers of abstraction and complexity to navigate.

Meemoo will make it possible for people to not only share such modules online, but also wire them together, experiment, and save output instantly online. This will lower the barrier to entry of creative coding by decreasing the number of applications needed to just one: the web browser. It will increase collaboration potential by allowing people to collaborate and share modifications all online.

3.5 WEB VIDEO REMIXER (2011), HTML

This project is the direct ancestor of Meemoo. I wanted to create a remixer for web video, including Youtube. The interface is split between controller and player, which are in separate HTML pages to allow the player page to be put on a separate monitor or projector.

There are several types of controllers that could be used to remix the video. Each video has triggers that corresponded to the keys of the keyboard. Patterns could be made by typing these trigger keys. Then the patterns could be put in order in the sequencer. It is a complex system, and I have not yet used all of the features to create a completed composition. This is a pitfall of tool design that I will be aware of in the future. Making these different kinds of music pattern sequencers into reconfigurable Meemoo modules will allow for more flexible tool design.

The controller page sends trigger signals as text to the player page using a JavaScript function⁶. When I was programming this in early 2011 this mechanism was only able to send text data. Around the middle of the year this limitation was lifted, allowing arbitrary data like images and audio to be sent. This expansion of the capabilities of browsers was a big inspiration to abstract this project from web *video* remixer to web *remixer* remixer.

4 Movie Player by voo2: <http://voo2.info/plugins-sources/voo2-movie-player-beta/>

5 Projection mapping / quad warping by Mehmet Akten: http://memo.tv/archive/projection_mapping_quad_warping_with_quartz_composer_vdmx

6 About `window.postMessage` from Mozilla: <https://developer.mozilla.org/en/DOM/window.postMessage>



Figure 18: Web Video Remixer

Meemoo is designed for hackability on all levels, from the interface to the source code. On the highest level, people can add and remove modules and reconfigure wires in the browser, without coding knowledge. On the next level, they can write and share their own modules with HTML, JavaScript and CSS that extend the capabilities of the framework. On the lowest level, the entire project is free software¹ under the MIT and AGPL licenses, which guarantee the right to fork the project and change how it works at any level.

I have experience with two dataflow visual programming environments: Quartz Composer and Pure Data. The feeling of direct manipulation and immediate feedback in working with these environments appealed to me. I was able to do graphics and audio processing with them that I could not do with any text-based coding environment. Therefore, I was able to explore new kinds of audio/visual experiments.

They are great tools for interactive installations, when you have control over the system. However, it is impossible to use them for creating web apps.

In the past two years browser capabilities have increased so that audio/visual programming is now possible with web standards. I realized that I could make my own visual programming environment with features that appealed to me from different paradigms: modularity, hackability, instant feedback, and shareability. Making a new creative tool is just a matter of wiring some modules together. I can write new modules in code that I am already comfortable with. Things made with this framework are easily shared online.

I hope that Meemoo might enable somebody to explore creative programming in the same way that my capabilities and imagination were extended with Quartz Composer.

*Modularity,
hackability, instant
feedback, and
shareability!*

4.1 MOZILLA WEBFWD

Development on Meemoo's ancestor project began in January 2011. In October 2011 Meemoo became a Mozilla WebFWD² fellow project. This program was created to mentor open-source projects that support The Mozilla Foundation's goals. One principle outlined in The Mozilla Manifesto that aligns particularly well with this project is: "Individuals must have the ability to shape their own experiences on the Internet" (The Mozilla Foundation, 2008).

¹ Meemoo source code is hosted on Github: <https://github.com/meemoo>

² Mozilla WebFWD: <https://webfwd.org/>

The WebFWD fellowship has given me connections to mentors in and around Mozilla, and weekly online seminars and group calls with all of the fellow projects. It is helpful to learn from the experiences of the other people involved in open-source projects. In December 2011 one representative from each project met at Mozilla Headquarters in Mountain View, California. There we had sessions with people involved in different levels of the Mozilla organization, from community relations to user experience³. It was a new and helpful experience to be around so many people that know more than me about web technologies.

4.2 SOFTWARE DESIGN FOR HACKABILITY

In open source projects, software design that considers potential collaborators makes code that is more hackable. I have studied the source code and design patterns of jQuery to make Meemoo more hackable on the code level.

4.2.1 *Common communication library for modules*

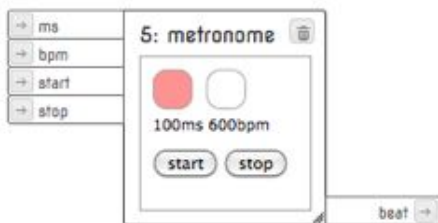


Figure 19: Module

In order to design Meemoo for open modular hackability it must be straightforward for other people to create modules. Meemoo modules are simple web pages that can be hosted anywhere. A module is referenced by its web address and loaded into the framework in an iframe. The framework surrounds the iframe with a resizable window with inputs on the left side at the top, and outputs on the right side on the bottom (Figure 19).

Each Meemoo module includes `meemoo.js`⁴, the JavaScript library that handles port defining and message routing. The inputs and outputs are then specified as in code sample C.1 on page 59. Because every module is including the same JavaScript library, it becomes a standard for message communication.

4.2.2 *Readable, sharable app source code*

The source code format for a Meemoo app is JSON (JavaScript Object Notation) which is fairly easy to read. This “text blob” stores the web address, position, connections, and state of all of the modules in the graph (code sample C.2 on page 60). Because it is a small amount of text, it is easy to share the app source code in email, forums, image descriptions, and comments.

³ Weekly seminar and summit videos: <https://webfwd.org/resources/webinars/index.html>

⁴ About meemoo.js: <https://github.com/meemoo/meemoo>

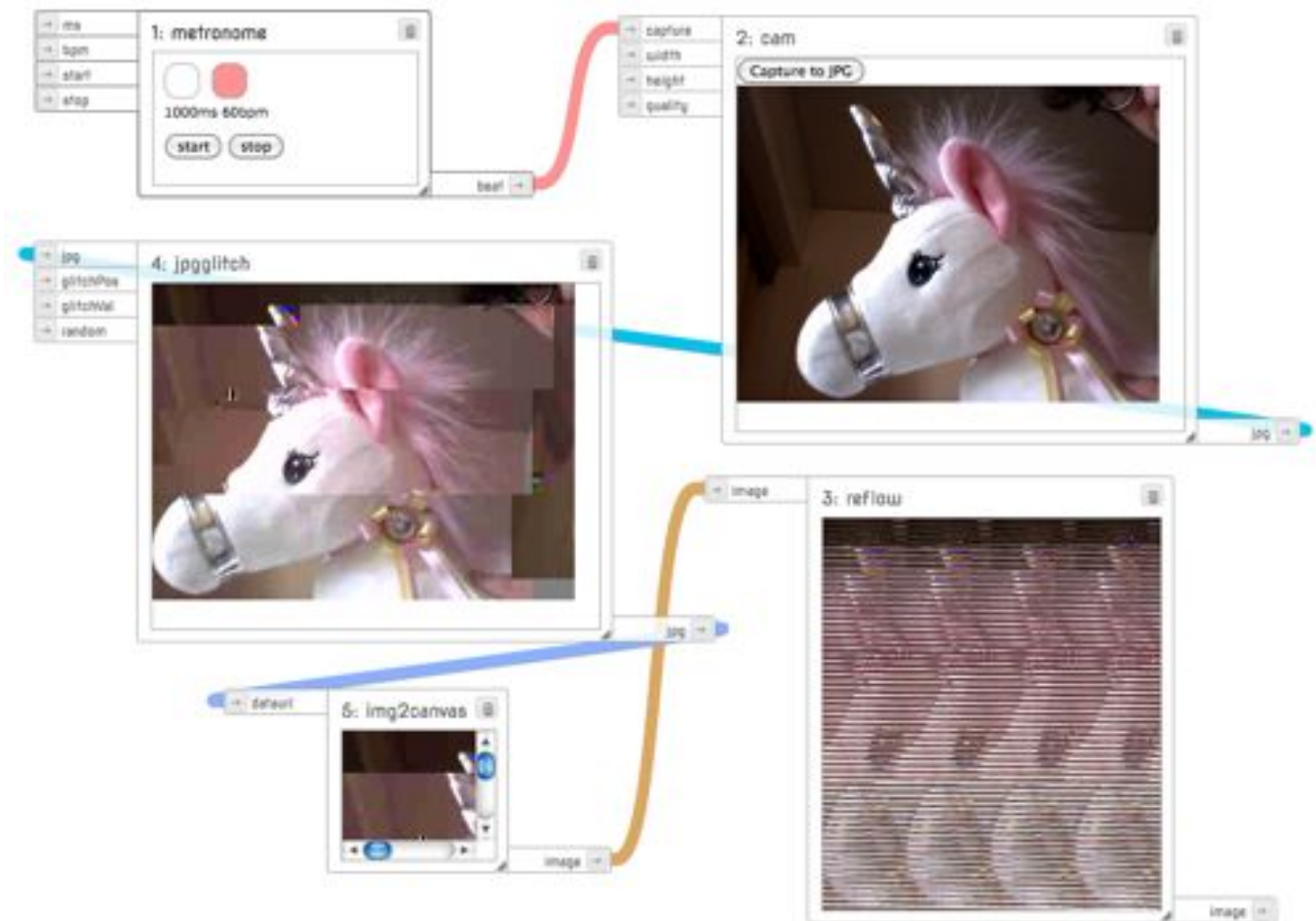


Figure 20: Glitch app

4.3 USABILITY DESIGN FOR HACKABILITY

The wiring view might be intimidating to some people at first glance. This is a tradeoff that every design makes, balancing simplicity and power. It is my hope that the simplicity of what is inside the modules will encourage people to later explore the power in the framework around the modules. This is called a layered or spiral approach to learning, where success with one level encourages exploration in the next (Shneiderman, 1986, p. 494).

4.3.1 *Direct manipulation*

One design difference between Meemoo and Quartz Composer is that Meemoo is able to have interface elements within each module. By way of example, an image filter module shows what it does to the image every time it receives image data (Figure 20). This should make it easier for people to understand how data flows through the application, and what each module does.

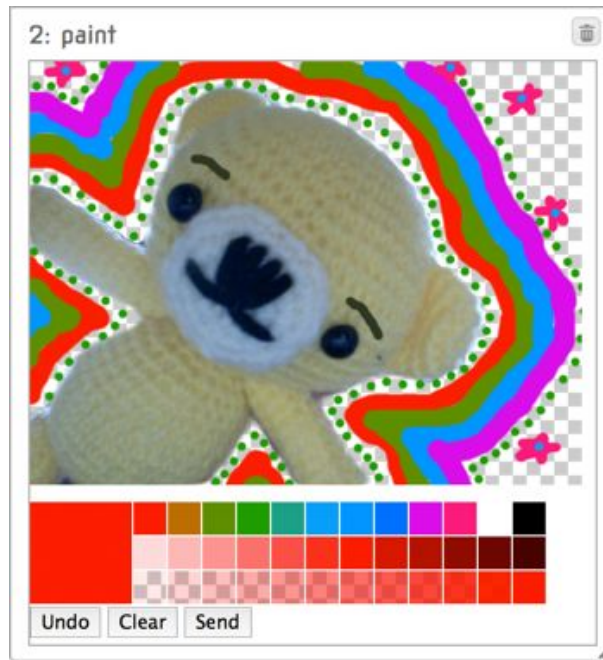


Figure 21: Paint module

4.3.2 *What is abstracted*

The Meemoo framework is a level of abstraction built on several levels of abstraction: JavaScript, browser, operating system, computer hardware. I could design modules that do only one logical programmatic function, which would lead to apps with more modules and wires to build the functionality required.

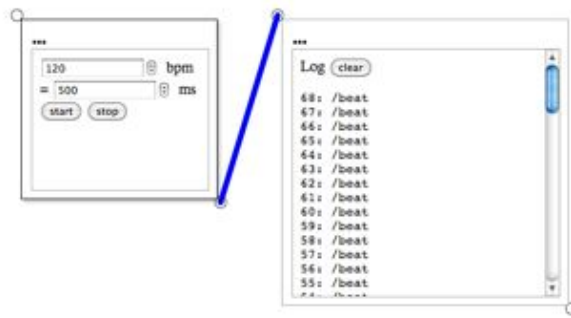
My preference is to abstract away more logic into each module. This makes each module a small self-contained application, like the paint module in Figure 21. This will make it easy for somebody to modify just one of the modules in an app. The framework dictates no preference to how much is abstracted into each module.

4.4 GRAPHIC DESIGN

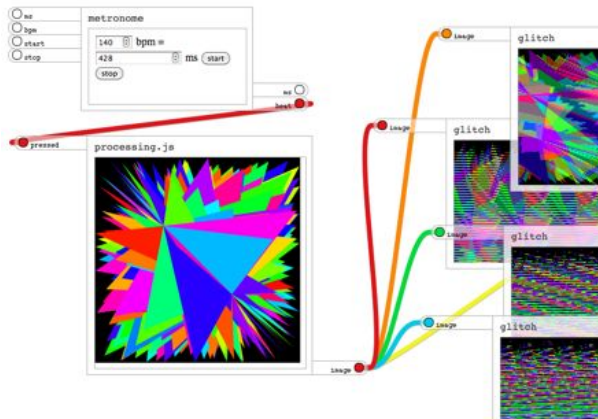
Pure Data has a quirky charm once you get over the initial steep learning curve, but some percentage of the charm is in being the only person on earth able to understand your own tangle of monochrome wires. One graphic design motivation was to make something nicer to look at and easier to understand at first glance than Pure Data. (Figure 22)

4.5 PERSONA AND SCENARIO-BASED DESIGN

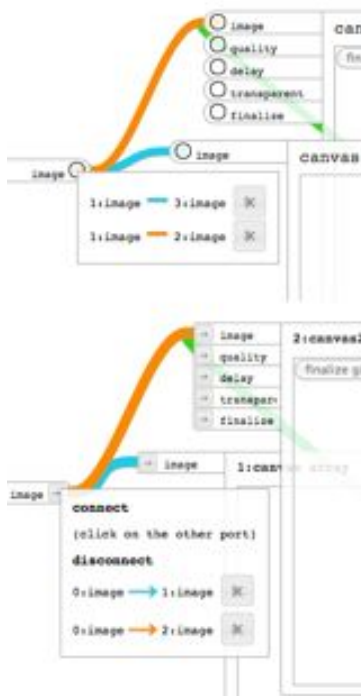
I used persona profiles and scenario-based design research to think about the potential audience for Meemoo. I used these methods as presented by Firefox user experience designer Jennifer Morrow at the



(a) September 2011, proof-of-concept



(b) November 2011, named ports and curved wires



(c) January 2012, making wires go to side of port



(d) January 2012, "Nova Flat" font

Figure 22: Graphic design evolution of Meemoo

Mozilla WebFWD Summit in December of 2011. I defined three persona profiles to describe people at different levels of engagement with Meemoo: the creator, the hacker, and the modder.

Creators will use Meemoo apps to make audio-visual media and share them online. *Hackers* will explore how the apps work, and rewire them to work differently. *Modders* will use web technologies to make and modify modules which will be used in new kinds of apps.

Meemoo is designed in a way that each of these levels leads to the next, encouraging people “down the rabbit hole” towards learning coding. I wrote a description of an imaginary person at each level of engagement, and how they would interact with the project (see Appendix B on page 57).

4.6 USABILITY TESTING AND FEEDBACK

In order to test the user experience of the framework, I did in-person talk-aloud sessions. This methodology was also presented at the WebFWD summit (Morrow, 2011). I had people interact with Meemoo, sometimes freely and sometimes with prompts or goals to accomplish. As they interacted with the system, I asked them to speak their thought process aloud as much as possible. This method of testing allowed me to “listen in” to their thought process, making it easy to identify points of frustration. I recorded the sessions with screen-capture software for future reference (Figure 23).

I tried to talk as little as possible, to see how people would interact with the framework if they had found it on their own. I reminded each person that I was not testing them, we were testing the usability of the project. I tested some people with no programming experience, and some people of varying experience with coding and visual programming. There were some usability issues that became quite obvious:

- An interactive introduction to the framework would be helpful to introduce the basic concepts and smooth the learning curve.
- I left the design of the modules plain to focus on the design of the framework. This seemed to make some people miss the buttons within the modules, and start by moving wires around. I will need to make the inside of the modules the primary focus so that people do not unwire the apps before they understand what they do.
- Modules can specify the kind of data that ports send and accept. I could implement something like the shaped variables from Scratch’s codeblocks to make it more obvious which ports can be wired together.
- Connecting wires by drag and drop was easy for all levels.



Figure 23: Usability testing



(a) Port information

(b) Tool-tip

Figure 24: Usability issues

- Disconnecting wires is currently done by a less direct method that proved much less intuitive. First the port must be clicked to show the port information popup in which there is a button with a scissor icon to disconnect the wire (Figure 24a). It was implemented this way as a temporary solution, and should be replaced with a direct method, more like pulling the wire out of the port.
- There is an indirect method to connect wires. The button for that also got in the way. The spacing between the “click” button and the title of the “disconnect” section made people put those items together. (Figure 24a)
- There is a method to input a value directly into the port from the port information popup. The submit function is a checkmark, which was overlooked by almost everybody. They tended instead to click on the button labeled “click,” which has a different function. (Figure 24a)
- Nobody saw the tool-tip information that shows when hovering over an element (Figure 24b). The browser takes too long to show the tool-tip for it to be an obvious way to find information.

This testing has made some usability issues quite obvious. As I work to design solutions for these issues I will continue this kind of testing.

APPS

A large part of the development of Meemoo was creating apps to demonstrate the framework. While creating these apps, I think of processes that would have general purpose coded as a module. Coding modules helps me think about features that could be added to the framework. Adding to the module library gives me more ideas about the kinds of apps that can be made.

I made two larger projects, an economic model and a VJ system. These two projects are quite different from each other. They show the breadth of possibility with this open framework.

5.1 EXAMPLE APPS

To demonstrate the different kinds of apps that can be created with Meemoo I have been making a collection of example apps¹. So far I have focused on simple animation tools, as this makes it easy to engage creatively with the framework. I made a module for instant uploading to Imgur, a popular image sharing website. The default image description provides a link back to the Meemoo app that created it. I am hoping that this will start leading people to the project, as I have not yet been actively promoting it to any online communities.

The three examples in Figure 25 illustrate the hackable nature of Meemoo apps. The first app is the simplest stop-motion app: from camera straight to looping animated GIF. The second app modifies the first by adding a paint module to allow doodling on each frame of animation. The third app modifies the second by moving one wire, thus making it a rotoscoping app for tracing images.

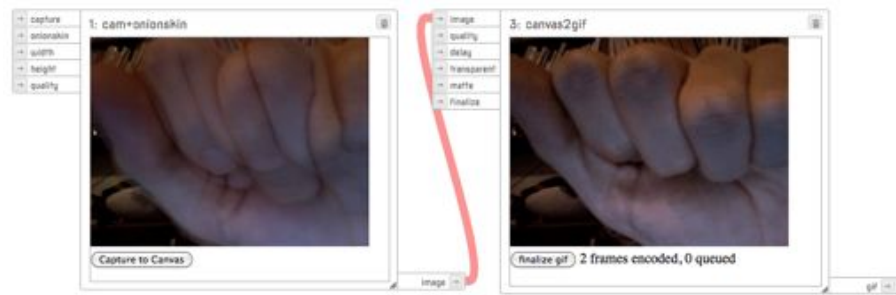
5.2 ECONOMIC SYSTEM MODELED WITH MEEMOO

“Just two days ago I clarified my thinking about the economic reform in Russia by programming a soft simulation of economic competition.” (Papert, 1993, p. 37)

I was amused to read this anecdote in Seymour Papert’s “The Children’s Machine,” as I had just done something quite similar. He used this example to show how his intellectual life has changed because of the availability of personal computers and their utility as thinking tools.

I had been imagining a system for online tipping that would make it possible for people to defer a certain percentage of a donation to

¹ Example apps: <http://meemoo.org/iframeframework/>



(a) Webcam to GIF



(b) Add a module: webcam to paint to GIF



(c) Change a wire: webcam to rotoscope tracing to GIF

Figure 25: Example apps

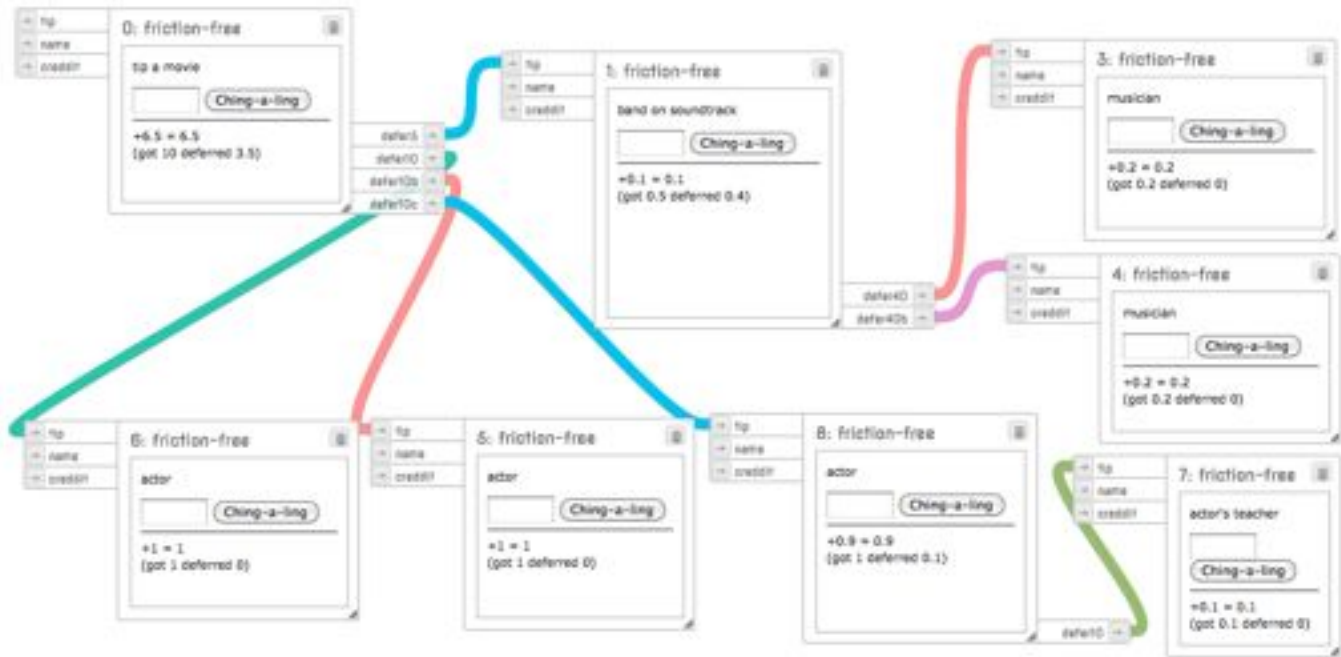


Figure 26: Economic model of deferred tipping

any number of other people. For example, a film collective makes their work freely available on Pirate Bay. I download the film, enjoy it, and decide to give them a tip to show my appreciation. I see a tip widget on their website that explains that 10% of tips are deferred to each of three actors, and 5% to a band on the soundtrack. I send them \$10, and the tip system divides and distributes the tip automatically (Figure 26).

These networks of tips could get quite complicated. The band's tips could be deferred to the band members. There could be two people that defer tips to each other (Figure 27).

In order to visualize these kinds of situations, I wrote a software simulation² using the Meemoo framework. To make this simulation I wrote a Meemoo module that, when a number is input, sends the percentages defined to each output and keeps the rest.

This application shows a nice feature of modular design. Each node in the graph is powered by the same module. This illustrates the object-oriented nature of the Meemoo framework, with modules that are like individual computers in a network that communicate with messages (Kay, 1996, p. 1).

Because this model is interactive software people can experiment with the system and see how tips to different nodes in the system get distributed. The model is built on Meemoo, so it is inherently hackable. If somebody disagrees with the model – or has a similar idea – they can fork the project and modify how it works to illustrate their point. This is an exciting possibility of making data visualizations and system models open source software.

² <http://meemoo.org/blog/2012-01-24-friction-free-post-scarcity-creative-economies/>

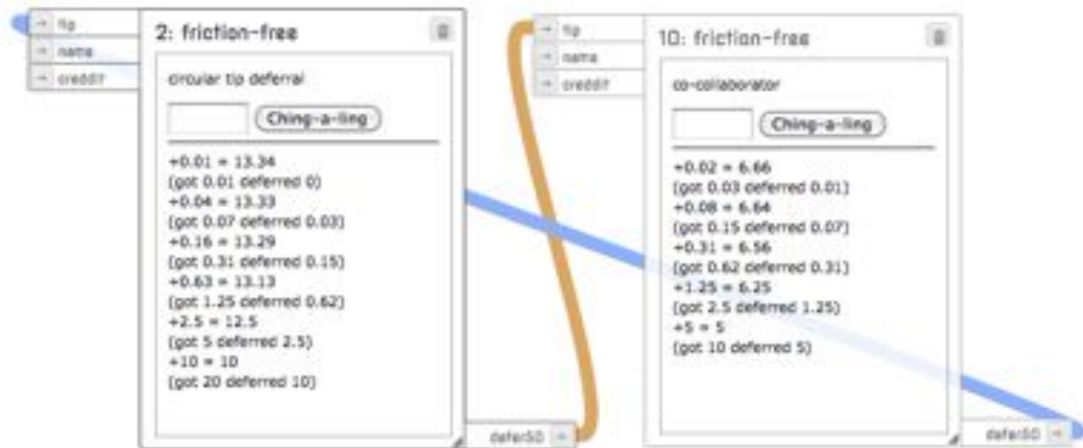


Figure 27: Circular tip deferral

5.3 LIVE ANIMATION VISUALS FOR DANCE PARTY

I contacted Ze Frank to ask if he would be a project advisor. He said that he was intrigued by the project, and gave me this warning:

“Creating ‘possibility spaces’ can be exciting for a number of reasons... but also can be a false God. It can be an excuse to never to actually grapple with whether there is value in the output itself, whether beauty is enough, whether people actually want what you are making...”

Partially thanks to this provocation, I decided to find a venue to perform with Meemoo. I was invited to VJ at Zodiac’s Side-Step dance festival club night on February 11, 2011. I used the gig as an opportunity to push Meemoo development and pressure-test the live-animation features.

For the gig I made some custom modules for creating a “world” into which I could insert animated sprites. On the software development side, I am happy that I decided to make two modules (Controller and World, Figure 28) share the same Backbone.js data model. Each module has its own view of the same data model, so the data passed through the wire has the same format on both sides.

As the party started and I was still coding furiously, adding features to the world module. Thirty minutes later the music tempo picked up, inviting people to the dance floor, and I made myself declare the coding done for the night. It was a thrill to see the first sprite hit the dance floor: multicolored glitter swirling in water.

We used clay and construction paper (and some glitter) as the basic building blocks of the visuals (Figure 29a). I am attracted to the textures and imperfections that come from using materials like these. Using the `taptempo` module, I synced the sprites’ animation to the beat of the music. It was fun to build these tiny animations and then throw them onto the screens around the dance floor (Figure 29d).

Juho Santasalo documented the performance³ in high definition video with a DSLR camera.

³ Video documentation of evening: http://youtu.be/T_tCyYGLWKM



Figure 28: “Meemoo World” interface for controlling animation

There are a few improvements and ideas that became clear in the course of the evening:

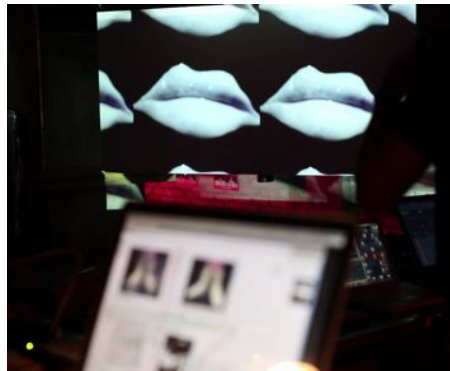
- Camera: I used a Sony Eyetoy webcam, which had unsatisfactory color reproduction. I chose art supplies with rich colors, but most of the color was washed out in the first step. Next time I will do some tests to find a better camera.
- Audience participation: I planned to use a Kinect depth camera to get silhouettes of people dancing into the world, but ran out of time. I was imagining using different animated textures for specified depth ranges.
- Flocking: I only had time to implement the tiled animation. The original concept was that sprites could be individual or flocks that would move around the screens.
- UX tweaks: Confirm dialog on every delete got annoying when juggling around modules. It would be better if there were an undo function, so delete could work without confirmation. A method for un/replugging wires that is more direct would make it quicker and easier to change the workings of the app.
- I made a hack to open the World module in a new window to view it fullscreen on the projectors. I plan on making this a built-in feature for any module.

Despite these limitations, I got good feedback about the visuals. People were interested in what I was doing, and came around to play with the art supplies. Doing dance party visuals powered by a web browser was a fun experiment. With some more development the browser limitations will be less aesthetically obvious. Performing under pressure was a good way to test the system.

Only *once* in the evening did a JavaScript warning pop up on the dance floor. I consider that a victory, and it made me laugh out loud when it happened.



(a) Claymation fun with art supplies



(b) Glitter lips



(c) Live wiring



(d) Two screens

Figure 29: Meemoo at Zodiak, photos by Juho Santasalo

FUTURE DEVELOPMENT

I believe that this idea is bigger than one developer and one master's thesis project. I hope to find resources to continue work, and to bring more people with varied talents into the project.

6.1 COMMUNITY FOR SHARING APPS

Meemoo was designed for sharing. I would like to take inspiration from elements of Reddit, Github, and an App Store to create a community for sharing and forking Meemoo apps. It will combine the community curation and subcommunities of Reddit, open source forking of Github, and the ease of use of an App Store. It will be relatively easy to make such a community scalable, because the source code of a Meemoo app is a small amount of text.

I imagine that a common point of entry to the community will be to see an image created with a Meemoo app. From there, you will be able to use the app that created the image. If you modify the app you can save it to your browser's local storage without signing in. If you decide to save it publicly, you can sign up for the community at that point. I would like to design the service where the first priority is to enable creative hacking. Roping the person into signing up for the community would be a secondary priority. (Figure 30)

An interesting possibility for a successful community will be tracing how modules and apps are forked through time. Each module can be considered the software manifestation of an idea, or meme. Ideas are recombinations of other ideas, and apps are recombinations of modules. That would make the source code of an app analogous to its DNA. The tree of forking and modification would be a direct map of the meme evolution of such a community.

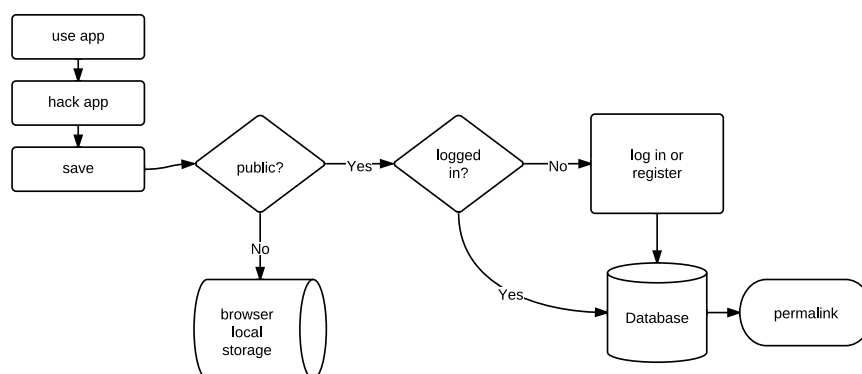


Figure 30: Flow for saving an app

6.1.1 *Socket communication*

A potential feature of a community infrastructure is socket communication. This would allow people to send data from an app on one device to a different app on another device. For example, I could send an image from a small camera app on my phone to a stop-motion animation app on my laptop. This will allow devices with limited capabilities to work in tandem with other devices.

This infrastructure could also be used to collaboratively edit Meemoo apps. This could have interesting uses in the classroom. For example, some students could be coding a game while other students are making the artwork for the game, with both groups able to see and test the results in real-time. This kind of collaboration happens in the online Scratch community, but due to software limitations it is not real-time (Resnick et al., 2009, p. 65).

6.2 TOUCHSCREEN SUPPORT

Meemoo has the potential to become a powerful tool for creative programming on touchscreen devices. The library of modules reduces the need to write code in a text editor.

Gestures for zooming and panning are common in touchscreen interaction, and could be an intuitive way of navigating a Meemoo application. Since Meemoo runs in the browser, zooming and panning already work smoothly. Module dragging and wire connecting will work soon.

Meemoo runs in browser, and in general JavaScript is slower than native code. However, as the power of devices and browsers increase, the kinds of apps that can be built with Meemoo will likewise increase.

High-resolution touchscreens are not the ultimate interactive interface. They do not make use of the range of motion and sensitivity of the hands and fingers, reducing all tactile interaction into the feeling of rubbing glass. Expanding the concept of visual programming to haptic/tactile programming could be a very interesting research direction.

6.3 CODE EDITING

Websites such as jsFiddle.net are designed to streamline the experience of editing and sharing web experiments and examples. The first screen of the site presents three text areas for HTML, CSS, and JavaScript, and a result frame that runs the web page output (Figure 31). This site is used by JavaScript library developers to demonstrate bugs. It is also used to make demonstrations for web programming questions and answers in forums such as StackOverflow. The imme-



Figure 31: jsFiddle

diacy of editing, saving, running, and sharing code in the browser makes this a time-saving tool for these communities.

I have been developing the first Meemoo modules using a traditional desktop code editor. I am used to switching between code editor and browser to test changes, and I probably make this mode switch hundreds of times per day. A code editor similar to jsFiddle that runs within Meemoo would eliminate the need to constantly switch applications to develop and test modules. Pressing “view source” on a module would open an editor showing the source code of the module. From this view, there would be options to edit the current module (if you have access to it) or fork the module into a new one.

The open-source programming environment DesignBlocksJS (Figure 32) uses Scratch’s codeblock programming metaphor. Combining these concepts, you could have the higher-level wiring hackability of Meemoo, and the lower-level coding hackability of codeblocks to edit the source of a Meemoo module. Programming modules with codeblocks would be a good transitional step between “hacker” and “modder” for beginning coders.

On touchscreens, coding with drag-and-drop codeblocks has the potential to be easier and more efficient than text-based coding. Some optimization would be required to account for the difference in mouse pointer and finger size.

6.4 TWENTY APPS TO BUILD WITH MEEMOO

In the spirit of Seymour Papert and Cynthia Solomon’s 1971 memo, “Twenty Things to Do With a Computer,” I present this list of potential Meemoo apps:

1. Instructional puzzle game based on rewiring modules
2. Kaleidoscope with reconfigurable mirrors
3. Experiment with video feedback with webcam pointed at screen

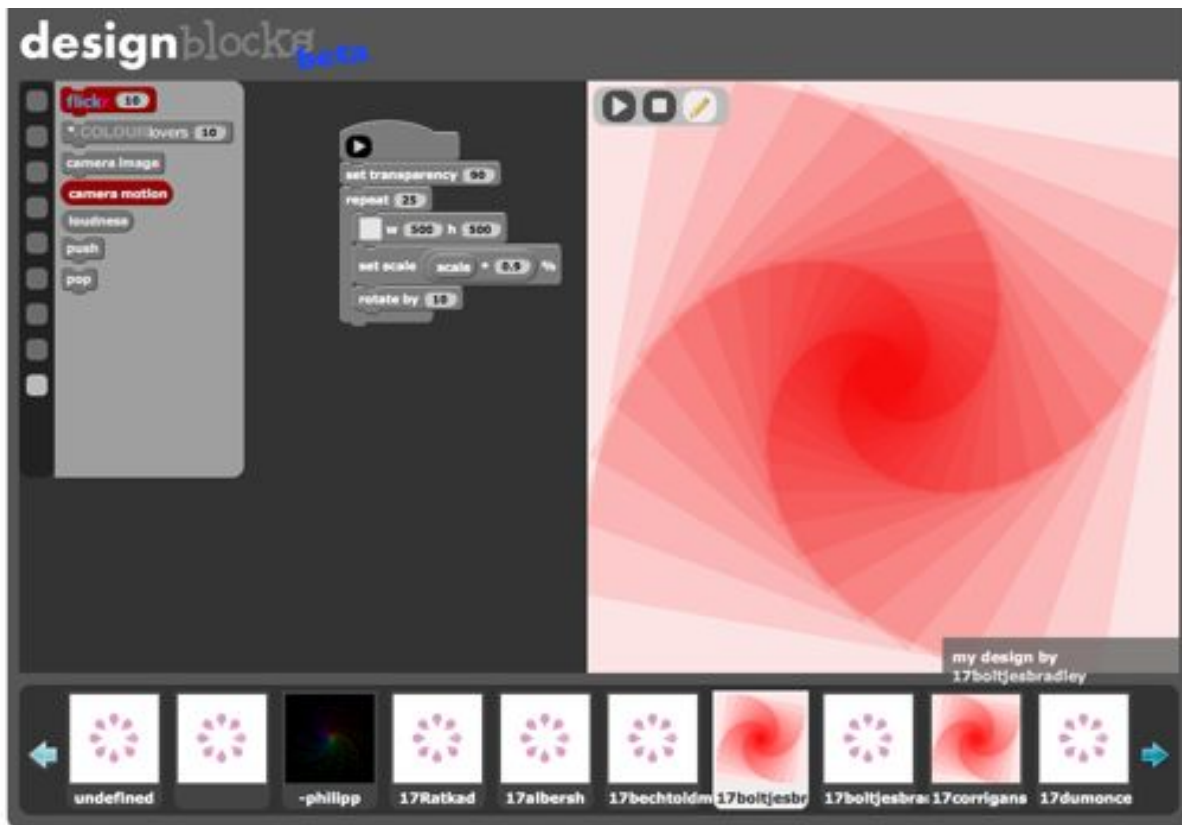


Figure 32: DesignBlocksJS

4. Collage that accepts images from many sources, like mobile phones
5. Text-to-song generator¹ with computer-synthesized voices singing in harmony
6. Artistic visualization of data from bio-sensors recorded while sleeping
7. Beatbox control of video mashup, where different sounds in the microphone trigger different video clips²
8. Hourglass module that flows virtual sand to other modules through the wires
9. Animated comic builder
10. Radiation data module to visualization and sonification modules³
11. Parallax-based 2.5D diorama builder
12. Arduino module to connect Meemoo apps to physical sensors
13. Two video players and a crossfader

¹ Speech synthesizer in 1K of JavaScript by Mathieu “po1” Henri: http://www.po1.org/releases/JS1K_Speech_Synthesizer/JS1K_Speech_Synthesizer.htm

² Idea from “sCrAmBlEd?HaCkZ!” by Sven König: <http://youtu.be/eRlhKaxcKpA>

³ Idea from “Radiation Always” by Ben Dromey: <https://vimeo.com/22329613>

14. (Animated) image macro generator
15. Side-scrolling game with placeholder image sprites that can be replaced
16. SVG pattern generator for laser-cut papercraft 3D model
17. Processing.js sound visualizer
18. TI-83 emulator⁴ to draw frames of geometric animation
19. Logo emulator⁵ that draws patterns with variables influenced by weather data
20. A Scratch game that draws different scenery based on location data

The last four examples show how – in the same way that computers and the Internet encompass all past and future media – a hackable creative coding environment that runs in the browser can encompass and interact with all other creative coding environments. The educational philosophies that developed these systems can be hacked, updated, and incorporated into new educational goals on the radically open web.

⁴ TI-83 in JavaScript by Cemetech & Kerm Martian:
<http://www.cemetechnet.net/projects/jstified/jstified.php>

⁵ Logo in JavaScript by Joshua Bell: <http://www.calormen.com/Logo/>

CONCLUSIONS

The performance portion of this thesis was an enjoyable learning experience. I would like to take what I learned and build on the live animation software developed. It would be great to collaborate with musicians and build a story with the visuals.

The main objective for the project portion of the thesis was to “design a modular dataflow visual programming framework using web technologies.” I think that this concept has been proven.

I will examine the secondary objectives:

- “The framework should afford non-coders the ability to modify creative web apps by configuring wires that represent how modules communicate.” According to usability testing, the framework does not yet satisfy this requirement for novices. I think that an interactive introduction to Meemoo would help soften its learning curve. There are also usability issues with the interface to address in the near future.
- “Apps created with the framework should have source code that is easy to read and share.” This has been accomplished by designing the source code to be a simple standard format. The complexity of each module’s code is abstracted to its web address, so the URL and the state of the module are all that is needed. The source code format of Meemoo apps is both concise and open-ended for future expansion.
- “Web coders should be able to extend the framework by creating modules using web standards. There should be a simple syntax to define the inputs and outputs of a module.” I am the only person that has written Meemoo modules, so I do not yet have any feedback from other people on this point. I designed the library to be simple to use for people with JavaScript experience.

I have made a framework that I look forward to building on for a long time. It is open-ended and the capabilities will increase with the capabilities of web browsers.

I have been designing this framework for myself, but with an eye to a potential future where others might find it valuable. I am interested to see what other people might build with it. I hope that it encourages other software designers to consider the hackability of their products.

BIBLIOGRAPHY

- Burroughs, William S. "The Cut-Up Method of Brion Gysin." *The New Media Reader*. eds. Noah Wardrip-Fruin and Nick Montfort. MIT Press, 1961. 90–91. (Cited on page 8.)
- Ellis, T. O., Heafner, J. F., and Sibley, W. L. "The GRAIL language and operations." (1969). URL http://www.rand.org/pubs/research_memoranda/RM6001.html?RM-6001 (Cited on page 13.)
- Gruber, John. "Choice Nuggets From Apple's Response to the FCC's Inquiry Regarding the Rejection and Removal of Google Voice Apps From the App Store." 2009. URL http://daringfireball.net/2009/08/apples_fcc_response (accessed on January 10, 2012) (Cited on page 6.)
- Hermes, Will. "All Rise for the National Anthem of Hip-Hop." 2006. URL <http://www.nytimes.com/2006/10/29/arts/music/29herm.html> (accessed on February 7, 2012) (Cited on page 8.)
- Himanen, Pekka. *The Hacker Ethic*. Random House Trade Paperbacks, 2001. (Cited on page 6.)
- Jobs, Steve. "Introducing the First iPhone at Macworld." 2007. URL <http://youtu.be/6uW-E496FXg> (accessed on February 17, 2012) (Cited on page 6.)
- Kay, Alan. "User Interface: A Personal View." *The Art of Human-Computer Interface Design*. ed. Brenda Laurel. Addison-Wesley, 1990. 191–207. (Cited on page 8.)
- . "The Early History of Smalltalk." *History of programming languages II*. ACM, 1996, 511–598. (Cited on pages 7, 10, 13, and 37.)
- Kay, Alan and Goldberg, Adele. "Personal Dynamic Media." *The New Media Reader*. eds. Noah Wardrip-Fruin and Nick Montfort, vol. 10. MIT Press, 1977. 393–404. (Cited on page 7.)
- Kushner, David. "The Making of Arduino." 2011. URL <http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino/0> (accessed on February 10, 2012) (Cited on page 17.)
- Manovich, Lev. "New Media from Borges to HTML." *The New Media Reader*. eds. Noah Wardrip-Fruin and Nick Montfort, vol. 41. MIT Press, 2003. 13–28. (Cited on page 8.)
- . *Software Takes Command*. 2008. URL <http://lab.softwarestudies.com/2008/11/softbook.html> (Cited on page 8.)

- Mister Jalopy, Torrone, Phillip, and Hill, Simon. "The Maker's Bill of Rights." *Make: Technology on your time* 4 (2005): 154. URL <http://makezine.com/04/ownyourown/> (Cited on page 16.)
- Morrison, J. Paul. *Flow-Based Programming: A New Approach to Application Development*. Createspace, 2010. (Cited on page 13.)
- Morrow, Jennifer "Boriss". "User Experience talk at Mozilla WebFWD Summit." 2011. URL <http://vid.ly/4o8d9l> (accessed on December 16, 20) (Cited on pages 30 and 32.)
- Netcraft. "March 2012 Web Server Survey." 2012. URL <http://news.netcraft.com/archives/2012/03/05/march-2012-web-server-survey.html> (accessed on March 10, 2012) (Cited on page 16.)
- O'Reilly, Tim. "What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software." 2005. URL <http://oreilly.com/web2/archive/what-is-web-20.html> (accessed on March 10, 2012) (Cited on page 18.)
- Papert, Seymour. *The Children's Machine: Rethinking School in the Age of the Computer*. Basic Books, 1993. (Cited on pages 10 and 35.)
- Papert, Seymour and Harel, Idit. "Situating Constructionism." *Constructionism*. Ablex, 1991. 1–11. (Cited on page 9.)
- Papert, Seymour and Solomon, Cynthia. "Twenty Things to Do With a Computer." *Massachusetts Institute of Technology A. I. Laboratory* (1971). (Cited on page 43.)
- Raymond, Eric S. "The Online Jargon File, version 4.4.8: Hacker." 2003. URL <http://catb.org/jargon/html/H/hacker.html> (accessed on February 10, 2012) (Cited on page 5.)
- Reas, Casey and Fry, Ben. *Processing: a programming handbook for visual designers and artists*. MIT Press, 2007. (Cited on page 11.)
- Resig, John. "Processing.js." 2008. (Cited on page 11.)
- Resnick, Mitchel, Maloney, John, Monroy-Hernández, Andrés, Rusk, Natalie, Eastmond, Evelyn, Brennan, Karen, Millner, Amon, Rosenbaum, Eric, Silver, Jay, and Silverman, Brian. "Scratch: Programming for All." *Communications of the ACM* 52 (2009).11: 60–67. URL <http://web.media.mit.edu/~mres/papers/Scratch-CACM-final.pdf> (Cited on pages 10 and 42.)
- Sadri, Pasha, Ho, Ed, Trevor, Jonathan, Cheng, Kevin, and Raffel, Daniel. "Introducing Pipes." 2007. URL <http://pipes.yqlblog.net/2007/02/07/introducing-pipes/> (accessed on March 7, 2012) (Cited on page 14.)
- Shneiderman, Ben. "Direct Manipulation." *The New Media Reader*. eds. Noah Wardrip-Fruin and Nick Montfort, vol. 97. Mendeley Ltd., 1986. 486–498. (Cited on page 29.)

- Stallman, R. "The GNU manifesto." 1985. URL <http://www.gnu.org/gnu/manifesto.html> (accessed on February 5, 2012) (Cited on page 15.)
- The Mozilla Foundation. "The Mozilla Manifesto." 2008. URL <https://www.mozilla.org/about/manifesto.en.html> (accessed on January 25, 2012) (Cited on page 27.)
- Victor, Bret. "Kill Math." 2011. URL <http://worrydream.com/KillMath/> (accessed on February 10, 2012) (Cited on page 1.)
- . "Inventing on Principle." 2012. URL <http://vimeo.com/36579366> (accessed on Feb 14, 2012) (Cited on page 8.)
- Villa, Luis. "The Libre Web Application Stack." 2010. URL <http://autonomo.us/2010/08/the-libre-web-application-stack/> (accessed on February 10, 2012) (Cited on page 16.)
- Vygotsky, Lev. "Play and its role in the Mental Development of the Child." 1933. URL <http://www.marxists.org/archive/vygotsky/works/1933/play.htm> (accessed on March 5, 2012) (Cited on page 18.)
- Wichary, Marcin. "GUI Gallery Guidebook: Guided Tour of Macintosh." 2005. URL <http://www.guidebookgallery.org/tutorials/mac1984/mousingaround> (accessed on February 10, 2012) (Cited on page 21.)
- Williams, Sam. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly Media, Inc., 2012. (Cited on page 16.)

APPENDIX



THE PITCH

When you think of an "app," do you think of something that you can open, hack, and change how it works? Meemoo wants to give you this freedom. If you can't open it, you don't own it. Meemoo is a framework that connects open-source modules, powered by any web technology. The way that the data flows from module to module is defined and visualized by colorful wires. If you can connect a video player to a TV, you can program a Meemoo app. Meemoo is:

- a web-based creative tool maker
- a layer between polished app and source code to make it easy for people to open, hack, bend, rewire, fork, and mod creative web applications
- an open-source framework that connects open-source modules
- like Arduino for the web
- 100% of the web
- focused on audio/visual interactive applications, but the potential uses are limitless

PERSONA PROFILES AND SCENARIOS

These profiles were written to have an imaginary person at different levels of engagement with the framework. They are fictional; any similarity to any person is coincidental.

B.1 CATE THE CREATOR

Cate is a nine-year-old girl who lives in the suburbs with her parents and thirteen-year-old brother. She likes using the video camera in her iPod to create movies starring her toys, narrating the story as she makes it up.

Her brother uses the Internet and Facebook every evening. Sometimes he shows her funny animated GIFs and videos that his friends link to. One of his friends linked to an animated GIF made with LEGO bricks. In the image description was a link to the stop-motion Meemoo app that made it.

They click on the link and see a Meemoo app with two modules wired together. The first module shows the image from their webcam. She clicks “capture” and a still capture jumps to the second module. She clicks “capture” again and the second module shows a two-frame looping animation of the two of them shifting a little bit back-and-forth, looped. They laugh at it, because her brother was making a funny face in one of the frames. They understand how it works now, so they clear the image and start over. They make an animation with more frames, making funny faces and jumping around the room. They save the resulting animated GIF image file.

Cate decides that it would be fun to make a short story with stop-motion animation. She gathers some toys and starts to plan the story.

B.2 HENRY THE HACKER

Henry is Cate’s older brother. His communication with his friends at school is peppered with references to Internet memes and inside jokes. He enjoys drawing comic book heroes in his notebooks.

After playing with Meemoo with his sister, Henry notices the “Add Module” feature. He looks at the list of modules that can be added to the app. He adds the paint module and sees that it is a simple version of a familiar paint application interface. He discovers how to add a connections between modules and wires the webcam module to the paint module. He takes a picture of one of his sister’s stuffed animal cats, and then paints a helmet on the cat’s image. He does this a couple more times, amused by the animated loop that he is creating.

He saves the GIF to Imgur, then posts the link on a friend's Facebook wall.

B.3 MOLLY THE MODDER

Molly is a 28-year-old programmer. She has been involved in the demo scene, writing code that pushes computers to make complicated real-time graphics. She got interested in an on-line challenge to make a demo that runs in the web browser using less than four kilobytes of JavaScript. Her entry was a flock of polygonal birds that moved around the screen according to a flocking algorithm.

Another entry in the contest made a simple drum synth sequencer. Molly thought it would be cool if the her birds could change the direction of their flight in rhythm with the drum demo. She added a little JavaScript to both demos to make them Meemoo modules and wired them together in the framework.

CODE SAMPLES

C.1 DEFINING INPUTS AND OUTPUTS (JAVASCRIPT)

This is an example Meemoo module with two functions: squaring numbers and reversing text. The inputs and outputs would be defined in Javascript like this:

```
Meemoo
  .setInfo({
    title: "example",
    author: "forresto",
    description: "this script makes a Meemoo module"
  })
  .addInputs({
    square: {
      action: function (n) {
        Meemoo.send("squared", n*n);
      },
      type: "number"
    },
    reverse: {
      action: function (s) {
        var reversed = s.split("").reverse().join("");
        Meemoo.send("reversed", reversed);
      },
      type: "string"
    }
  })
  .addOutputs({
    squared: {
      type: "number"
    },
    reversed: {
      type: "string"
    }
  });
```

C.2 MEEMOO APP SOURCE CODE (JSON)

This is the source code of the app shown in Figure 1 on page 2. There are two modules (nodes) connected with one wire (edge). A webcam module sends an image to a painting module¹.

```
{
  "info": {
    "title": "cam doodle",
    "author": "forresto",
    "description": "webcam to doodle"
  },
  "nodes": [
    {
      "src": "http://forresto.github.com/meemoo-
        camcanvas/onionskin.html",
      "x": 127, "y": 57, "z": 0, "w": 342, "h": 283,
      "state": {
        "quality": 75,
        "width": 320,
        "height": 240
      },
      "id": 1
    },
    {
      "src": "http://forresto.github.com/meemoo-paint/
        paint.html",
      "x": 634, "y": 53, "z": 0, "w": 377, "h": 342,
      "state": {
        "linewidth": 2
      },
      "id": 2
    }
  ],
  "edges": [
    {
      "source": [ 1, "image" ],
      "target": [ 2, "image" ]
    }
  ]
}
```

¹ Try this app: <http://meemoo.org/iframe/#/example/camdoodle>



<http://meemoo.org/>

Copyright © 2012 Forrest Oliphant
Some Rights Reserved

“Meemoo: Hackable Web App Framework” by Forrest Oliphant is licensed under a
Creative Commons Attribution-NonCommercial 3.0 License.

<http://creativecommons.org/licenses/by-nc/3.0/>