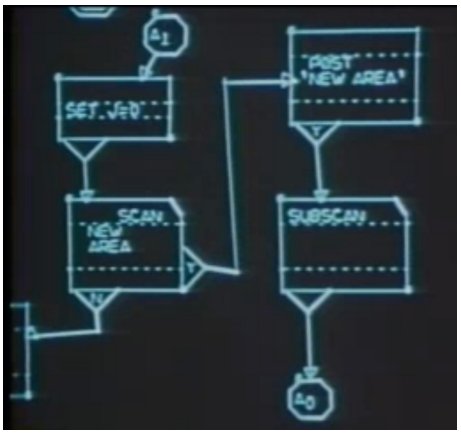
# MEEMOO: CREATIVE HACKABLE WEB APP FRAMEWORK

FORREST OLIPHANT




Design for Hackability

March 2012 – version 0.9 DRAFT

ABSTRACT

In this thesis I describe...

ACKNOWLEDGEMENTS

LeGroup (Learning Environments research group) in Media Lab Helsinki. Finland.
Mozilla WebFWD.
Aino and Ilo.

# CONTENTS


1 INTRODUCTION 1

2 CONTEXT 3

2.1 Hackers and Hackability 3

2.1.1 iOS 4

2.2 Metamedium 5

2.3 Tools 6

2.3.1 Direct manipulation 7

2.4 Programming for children (and other artists) 7

2.4.1 Logo (1967) 7

2.4.2 Smalltalk (1972) 8

2.4.3 Scratch (2006) 9

2.4.4 Processing (2001) 10

2.5 Visual programming languages 11

2.5.1 GRAIL (1969) 11

2.5.2 Audio/visual environments 11

2.5.3 On the web 11

2.6 Free Software Movement 12

2.7 Maker Movement 13

2.8 Plugin architecture 13

2.8.1 jQuery 13

2.8.2 Arduino 15

2.9 The Internet 16

2.9.1 Participatory media online 16

3 PREVIOUS WORK AND MOTIVATION 19

3.1 Early experiences 19

3.2 Voice Synthesizer (2002), Flash 20

3.3 Megacam (2010), Flash 21

3.4 Opera projection mapping (2011), Quartz Composer 21

3.5 Web Video Remixer (2011), HTML 22

4 DEVELOPMENT 25

4.1 Software design for hackability 25

4.1.1 Common communication library for modules 26

4.1.2 Readable, sharable app source code 26

4.2 User experience design for hackability 26

4.2.1 Direct manipulation 26

4.2.2 Visual programming "patching" metaphor 26

4.2.3 What is abstracted 27

4.3 Graphic design 27

4.4 Persona and scenario-based design 27

5 TESTS AND RESULTS 29

5.1 Usability testing and feedback 29

5.2 Economic model illustrated with Meemoo 30

5.3 Live animation visuals for dance party 30
6 FUTURE DEVELOPMENT 33
6.1 Community for sharing apps 33
6.2 Touchscreen support 33
6.3 Code editing 33
6.4 Socket communication 35
6.5 Meemoo hardware 35
6.6 Twenty Apps to Build With Meemoo 35
7 CONCLUSIONS 37
Bibliography 39

APPENDIX 43
A PERSONA PROFILES AND SCENARIOS 45
A.1 Cate the Creator 45
A.2 Henry the Hacker 45
A.3 Molly the Modder 46
B CODE SAMPLES 47
B.1 Defining Inputs and Outputs (JavaScript) 47
B.2 Meemoo App Source Code (JSON) 48

# 1 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 tools (see 2.2 on page 5). Learning traditional computer programming still takes a major commitment, and 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. People that create tools that are open to modification afford other people the ability to learn from their tools and 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 realized that I could 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 have designed and created a web-based 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. There should be a simple syntax to define the inputs and outputs of a module.
- Apps created with the framework should have source code that is easy to read and share.
- Web developers should be able create new modules for the framework using a simple standard.

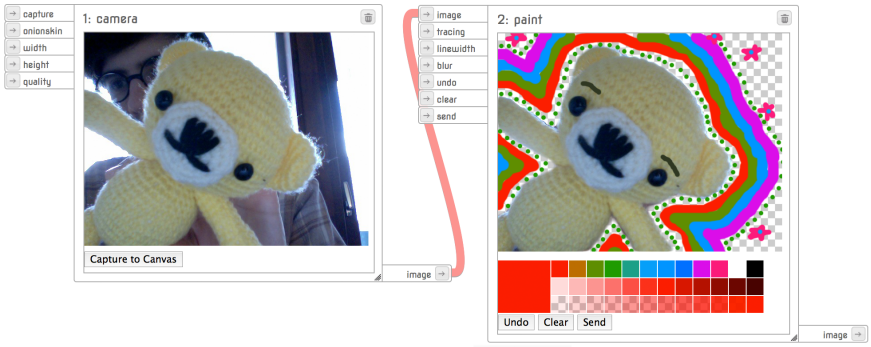



Figure 1: Meemoo screen-shot (March 2012)

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 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. Modifying and making new apps is done in the same browser interface, so no external developer tools are needed.

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

So far I have focused module development on realtime animation tools, as this makes it simple to explain 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 many 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 Meemoo[1], which is a framework for hackable web apps. It is also a performance[2] with an app created with the Meemoo framework for live animation (see 5.3 on page 30).

1 Demos and source code available at http://meemoo.org/
2 Live animation performance documentation: http://youtu.be/T_tCyYGLWKM

# 2 CONTEXT

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

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* instead of *users* and *developers*, as I would like to blur this distinction with the project.

## 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.

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

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. 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 Hackasaurus[2], both aimed at getting children to explore coding.

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 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 between the designer and user roles are well-defined in this way.

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

1 Hackity Hack!: http://hackety.com/
2 Mozilla Hackasaurus: http://hackasaurus.org/

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. Garageband[3] and Photoshop Touch[4] 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 don't afford any scripting-based hackability, and can only communicate with services that the designers have chosen.

There are some apps that do allow some programming-based hackability: apps that are code editors. These include Codea[5] by Two Lives Left for Lua, Processing.js Mini-IDE[6] by Brian Jepson, and GLSL Studio[7] by kode80 for OpenGL shaders. These three apps are development environments that deal with the affordances and constraints of writing code on touchscreen devices in different ways. For example, Codea uses the affordances of touch-screen interaction to manipulate variables with widgets embedded in the code, like popup number sliders and color pickers.

However, without an external keyboard, any kind of extended writing on touchscreen devices 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 9) and modular dataflow programming, like Meemoo. These devices are powerful computers, and they should be easily programmable to keep the full potential of the metamedium.

## 2.2 METAMEDIUM

The Dynabook was a research project of Xerox Palo Alto Research Center in 1968. It was a concept design that 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

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
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://glslstudio.com/

> 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 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)

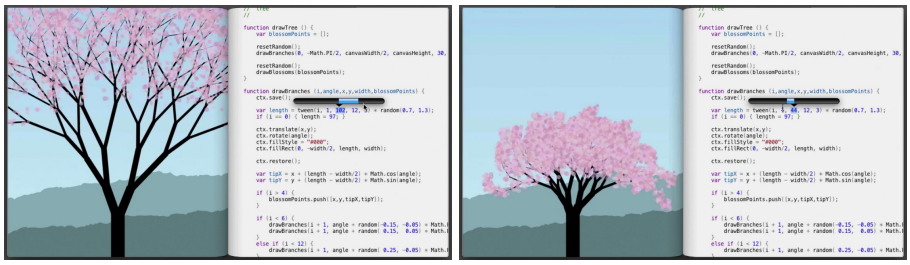
In order for the full potential of this metamedium to be realized, people should be able to create and modify their own tools.

## 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 lead 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.



(a) (b)

Figure 3: Bret Victor's direct manipulation of code

> "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)

### 2.3.1 *Direct manipulation*

Ben Shneiderman outlines

...

...

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.

## 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 build knowledge structures through experiences. 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 a 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).

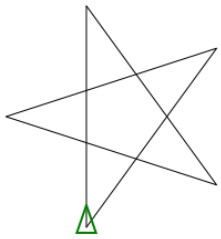


Figure 4: Logo turtle drawing a star

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 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 their own 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 different 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)

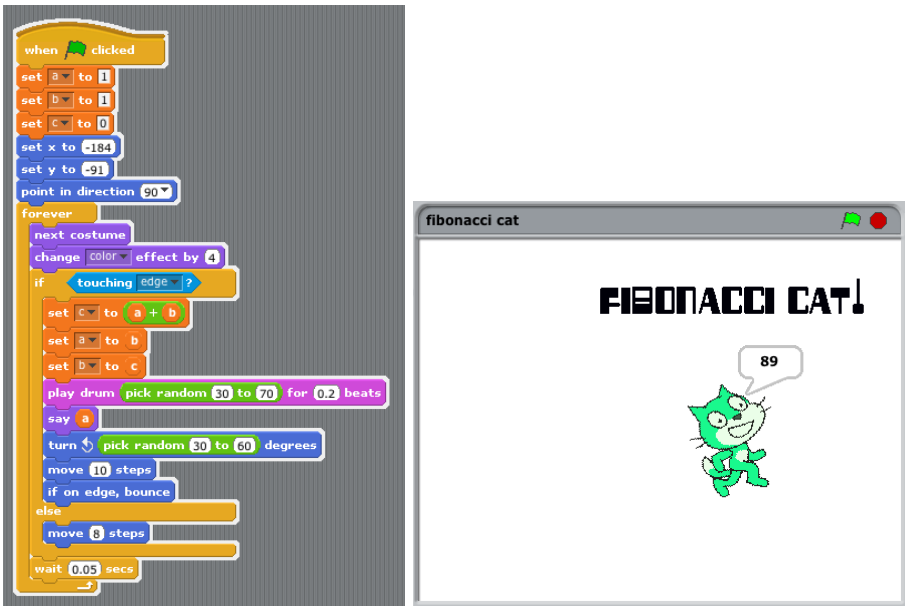
Although creating a script with code blocks is more like snapping Legos together than writing code, it is still coding. The shape of the control structures is a direct metaphor to how code syntax works, and more visually obvious than brackets or indentation. I imagine those logical structures are transferable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

I feel a little cheated to not have had Scratch when I was a child. I would have loved it. I extended my graphing calculator scripting experiments from high school with an absurdist animation of a cat running into walls and reciting Fibonacci numbers[8] (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.

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




(a) Script (b) Output

Figure 5: Fibonacci Cat! Scratch program by the author

Code can be changed as the project is running, and the results change immediately[9]. This brings direct manipulation to coding, in that changes in the code can be seen in real-time, without recompiling or refreshing. This would be a valuable feature for any programming environment, and it is especially 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).

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 logical structures. I would recommend anybody teaching programming to beginners to start with Scratch, at least for the first lesson.

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

## 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. "It was direct manipulation, it was analogical, it was modeless, it was beautiful." (Kay, 1996, p. 10)

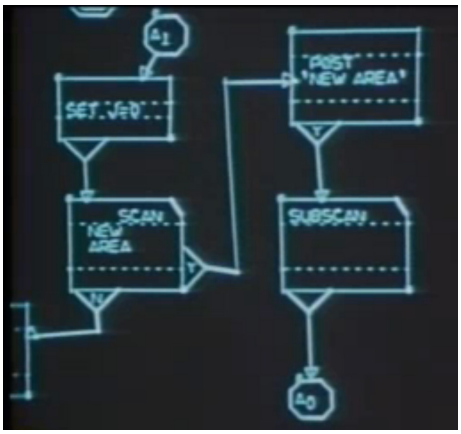



Figure 6: GRAIL

### 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), 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 much easier to learn a system based on this metaphor than to learn a system based on linear-textual coding.

### 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 "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.

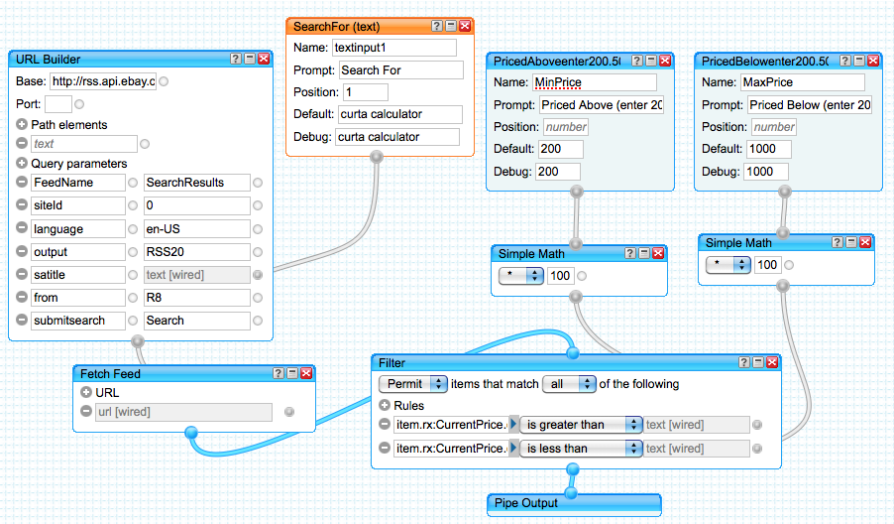



Figure 7: Yahoo! Pipes layout that watches eBay for items with a search term and price range

The open modular design of Meemoo allows its capabilities to keep pace with the capabilities of web browsers.

## 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, jQuery[10] and Backbone.js[11]. 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 complex maintainable web application.

10 jQuery: http://jquery.com/
11 Backbone.js: http://documentcloud.github.com/backbone/

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 wouldn't 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 "not reinventing the wheel" and "standing on the shoulders of giants" (Williams, 2012, p. 138).

It is not just the JavaScript libraries that Meemoo is built on that are free and open-source. 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 ever" 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" or app view is easy to open and see the wiring view. I'm planning on using a screw icon (Figure 8) for all components that you can "open" and view the source. The wiring view is analogous to an electronics schematic.

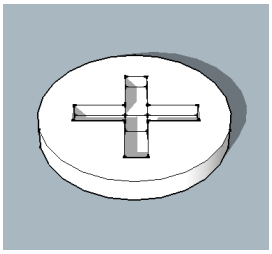


Figure 8: Open me!

## 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

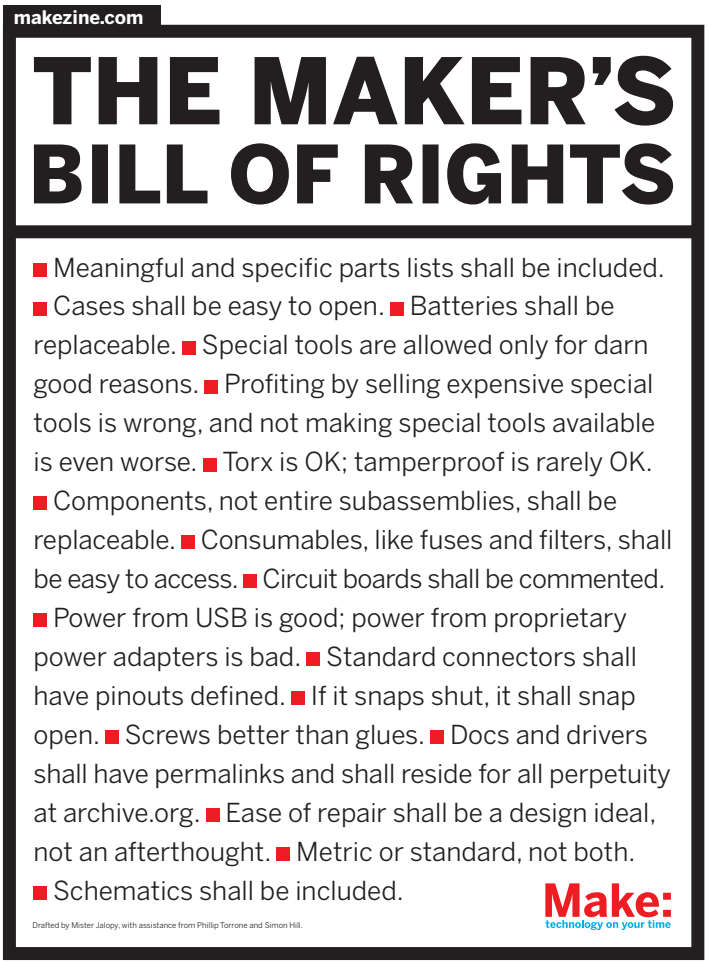



Figure 9: The Maker's Bill of Rights

site dedicated to jQuery[12] claims to have over 2000 listed in their directory. The only JQuery plugin that I'm using for Meemoo is the official JqueryUI[13], 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*

The Arduino was designed 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. They designed it 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. The creators say that the most important impact of the Arduino is "the democratization of engineering." (Kushner, 2011)

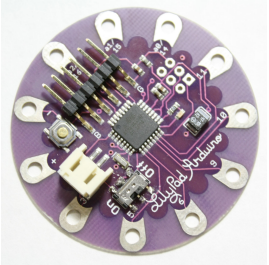


Figure 10: Lilypad Arduino

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 Gameduino[14] and Pulse Sensor[15]. (Both of these projects were funded on Kickstarter and raised many 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 with other modules through that framework. In the same way, 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 make and share their own modules.

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

## 2.9 THE INTERNET

People learn by imitating and communicating with people with more skill. Lev Vygotsky refers to the difference between what can do independently and what we can do with guidance the "zone of proximal development." This theoretical space is where learning happens for children, and can be seen both in play and in the classroom. (Vygotsky, 1933)

The Internet provides a social communication framework where people can learn from others: the ultimate zone of proximal development. Like most of my programming projects, Meemoo has pushed me into this zone with several challenges, stretching my capabilities as a programmer. Stack Overflow[16] has been an especially helpful community for learning from the questions and answers of others. The design of these platforms influences the kinds communities that form around them.

### 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.

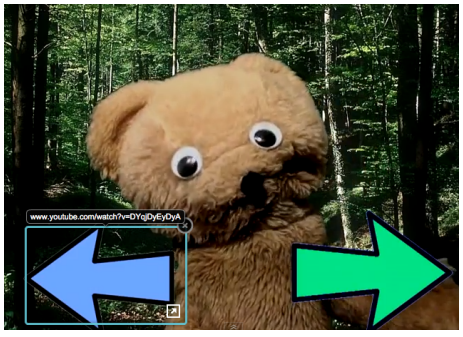



Figure 11: The Gilady Land Interactive Story

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 make 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 sytem. In the

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

17 "The Gilady Land Interactive Story" http://youtu.be/spVMyoUcuR4

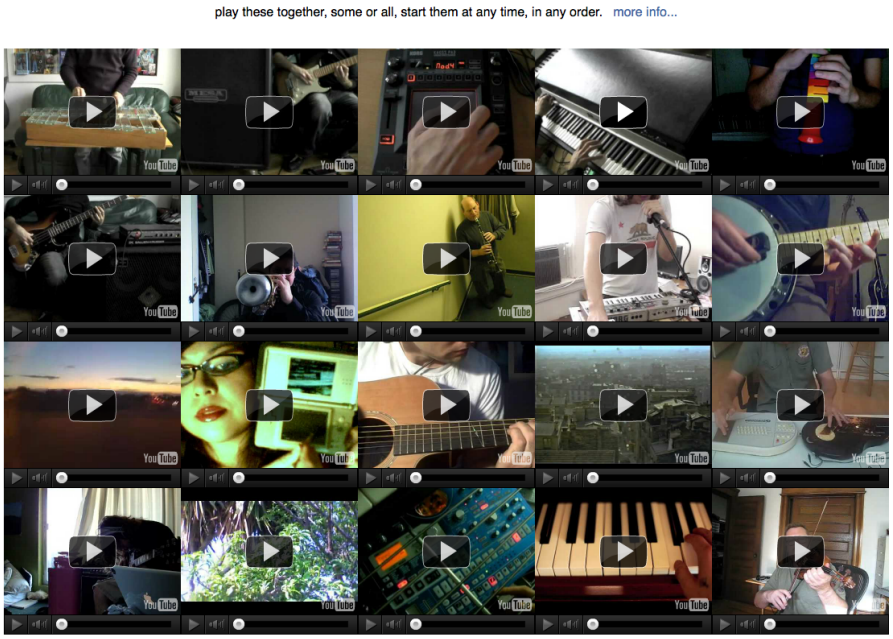



Figure 13: In B Flat

end of each section of the story, viewers are presented with two choices as hyperlinks within the video (Figure 11). 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 12).

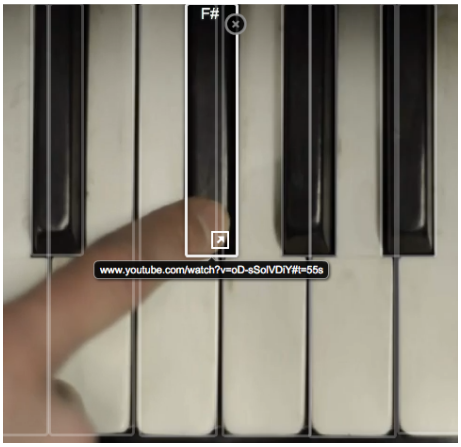



Figure 12: Play the Piano

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 one HTML 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

18 "Play the piano" http://youtu.be/oD-sSolVDiY
19 "In B Flat" http://www.inbflat.net/

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 API[21] has enabled developers to create applications that can access and edit Flickr's massive database of photos and metadata[22]. These APIs increase the web's hackability, but only for people with coding skills.

Meemoo is a web app that use and extend 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/

# 3

# 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 this thesis project: 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.

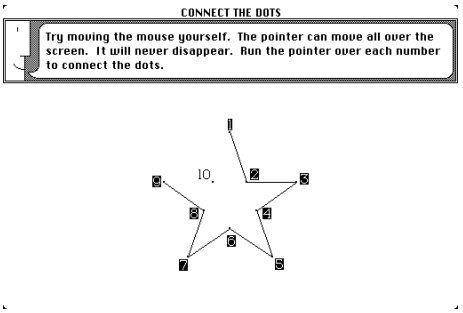



Figure 14: Mousing Around

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 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 a lot 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.

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 size and 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 me.

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 ambitious programming that I had done at the time. The result was a voice "synthesizer"[1] that used looped samples

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

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).

In the original concept, I 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.


Figure 15: Voice Synthesizer

## 3.3 MEGACAM (2010), FLASH

Inspired in part by Lomo cameras, I made a series of webcam toys in Flash[2]. 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.

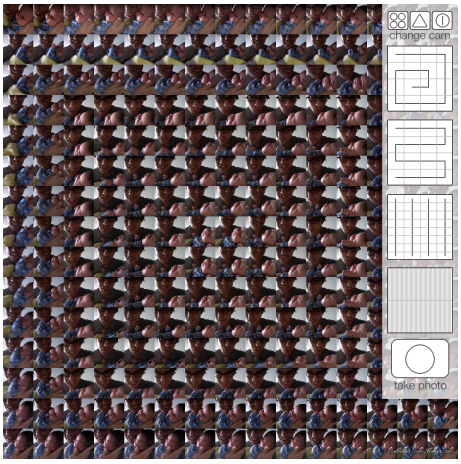



Figure 16: Megacam

## 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 Opera. I found Quartz Composer modules for midi control, video playback[3], and projection mapping[4]. 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.

2 Megacam: http://sembiki.com/megacam/
3 Movie Player by vo02: http://vo02.info/plugins-sources/vo02-movie-player-beta/
4 Projection mapping / quad warping by Mehmet Akten: http://memo.tv/archive/projection_mapping_quad_warping_with_quartz_composer_vdmx

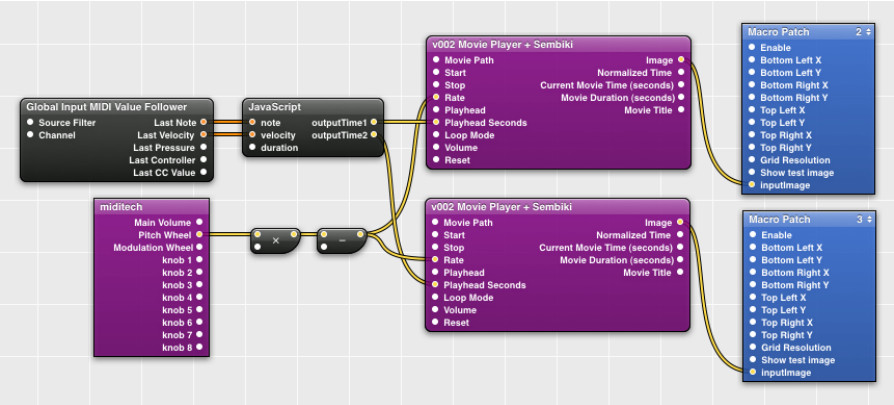



Figure 17: Connecting open-source modules in Quartz Composer


Figure 18: Web Video Remixer

Developing this application together 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 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 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 was split between controller and player, which were in separate HTML pages to

allow the player page to be put on a separate monitor or projector. The controller page sent trigger signals as text to the player page using the same mechanism as the current Meemoo uses to send data from module to module. In early 2011 this mechanism, `window.postMessage`[5], was only able to send text data to another window. 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.

There were several types of controllers that could be used to remix the video. Each video had 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 never got around to actually using it. This is a pitfall of tool design that I would like to avoid in the future. Making these different kinds of music pattern sequencers into reconfigurable Meemoo modules will allow for more flexible tool design.

5 About `window.postMessage` from Mozilla: https://developer.mozilla.org/en/DOM/window.postMessage

# 4

# DEVELOPMENT

Development on Meemoo's ancestor project began in January 2011. In October 2011 Meemoo became a Mozilla WebFWD[1] 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).

The WebFWD fellowship has given me connections to mentors in and around Mozilla, and weekly online seminars. We also have a weekly call with all of the projects. It is helpful to learn from the experiences of the other projects. In December 2011 one representative from each project met at Mozilla Headquarters in Mountain View, California. It was a new experience to be around so many people that know more than me about web technologies.

## 4.1 SOFTWARE DESIGN FOR HACKABILITY

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 lowest level, the entire project is free software[2] 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, so 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 and JavaScript engines have been made fast enough 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

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

1 Mozilla WebFWD: https://webfwd.org/

2 Meemoo source code is hosted on Github: https://github.com/meemoo

matter of wiring some modules together. I can write new modules in code that I am already comfortable with. Things made with this toolmaker 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.

### 4.1.1 *Common communication library for modules*

In order to design Meemoo for open modular hackability it must be easy for other people to create modules (see 2.8 on page 13). 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.

Each Meemoo module includes meemoo.js[3], a JavaScript library which handles message routing. The inputs and outputs are then specified as in code sample B.1 on page 47. Because every module is including the same JavaScript library, it becomes a standard for message communication.

### 4.1.2 *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 B.2 on page 48). Because it is a small amount of text, it is easy to share the app source code in email, forums, image descriptions, comments, etc.

## 4.2 USER EXPERIENCE DESIGN FOR HACKABILITY

### 4.2.1 *Direct manipulation*

Ben Shneiderman (Shneiderman, 1986)

Visual indication of what is happening in each module. (Like TouchDesigner). Dragging to change variables.

### 4.2.2 *Visual programming "patching" metaphor*

"The use of flexible cords with plugs at their ends and sockets (jacks) to make temporary connections dates back to manually operated telephone switchboards."

3 Meemoo.js: https://github.com/meemoo/meemoo

### 4.2.3 *What is abstracted*

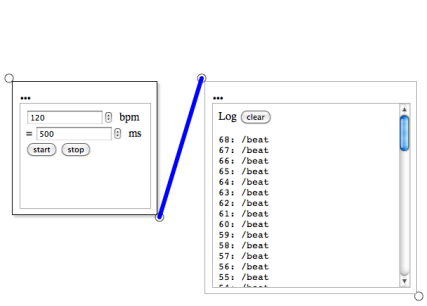
## 4.3 GRAPHIC DESIGN

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 19)

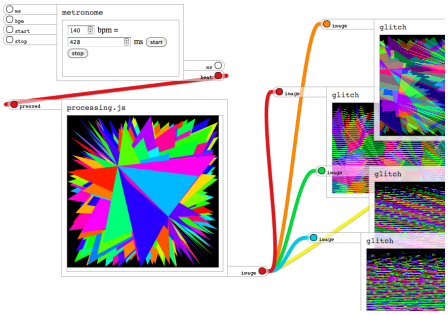
## 4.4 PERSONA AND SCENARIO-BASED DESIGN

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

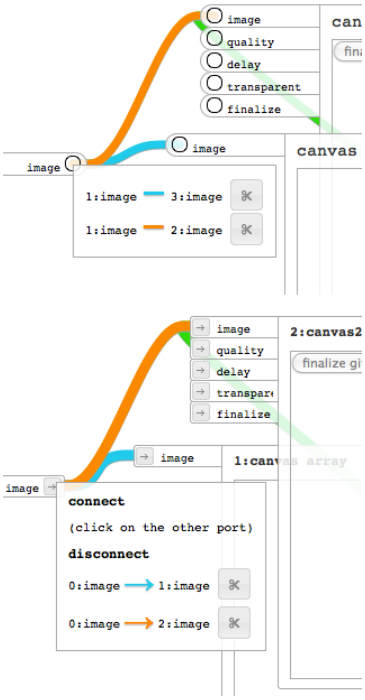
*Creators* will use Meemoo apps to make audio-visual media and share them online. *Hackers* will explore how the apps work, and rewire them to work differently. *Modders* will use web technologies to modify modules and create new modules which will be used in different kinds of apps. It is designed in 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 A on page 45).



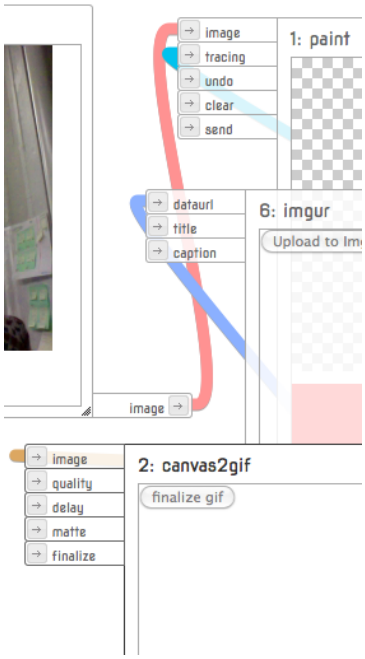

(a) September 2011, proof-of-concept




(b) November 2011, named ports and curved wires




(c) January 2012, making wires go to side of port instead of behind; port information pop-up




(d) January 2012, "Nova Flat" font

Figure 19: Graphic design evolution of Meemoo

# 5 TESTS AND RESULTS

## 5.1 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 Web-FWD 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 the thought process of the person using the system, making it easy to identify points of frustration.

I tried to talk and react as little as possible, to see how people would interact with the framework if they had found it on their own. I reminded everybody that we were not testing them, we were testing the usability of the project.

...

I recorded the sessions with screen-capture software for future reference (Figure 20).

There were eight sessions of 10-20 minutes each.

Some of the points of

Aino - Camdoodle

Ginger - "You should add an onionskin"

Teemu - Metronome animation

Jona - "Can I use this in my class?"

Facebook Beta group

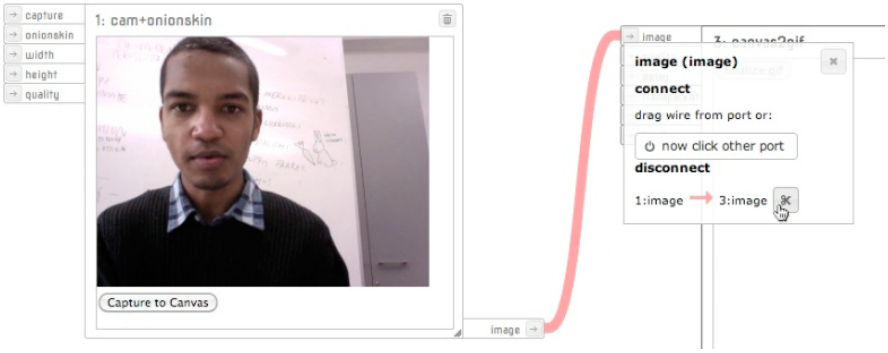



Figure 20: Usability testing

## 5.2 ECONOMIC MODEL ILLUSTRATED WITH MEEMOO

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

## 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 idea, but 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 in a live setting. I was invited to do visuals for a Zodiak's Side-Step dance festival club night. I used the gig as an opportunity to push Meemoo development and pressure-test the live-animation features.

For the gig I made some special 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) share the same Backbone model. Each module has its own view of the same model, so the data passed through the wire will be the same 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. 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 music's beat. It was fun to build these tiny animations and then throw them onto the screens around the dance floor. [1]

There are lots of improvements and ideas that came up in 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.

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

Figure 21: Meemoo at Zodiak

- Audience participation: I planned to use Kinect 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. 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 a lot of 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, and with a few more display options I think that the limitations would have been 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 a lot when it happened.

# 6 FUTURE DEVELOPMENT

This idea is bigger than one developer and one master's thesis. I plan on finding 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 elements of an App Store, Reddit, and Github to create a community for sharing and forking Meemoo apps. Because of the small amount of source code to describe a Meemoo app (Algorithm **??**) it will be relatively easy to make the community scalable.

## 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 take some more development time.

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

## 6.3 CODE EDITING

Websites such as jsFiddle.net are designed to streamline the experience of editing and sharing HTML 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 22). This site is used by JavaScript library developers to demonstrate bugs, as well as to make demonstrations for web programming questions and answers in forums such as StackOverflow. The immediacy 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

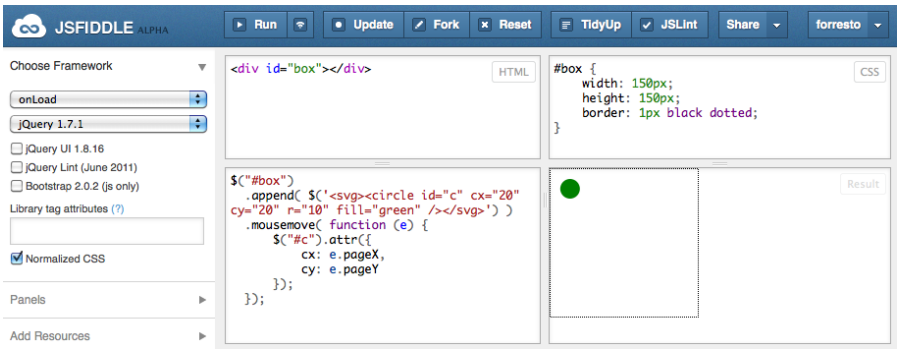



Figure 22: jsFiddle

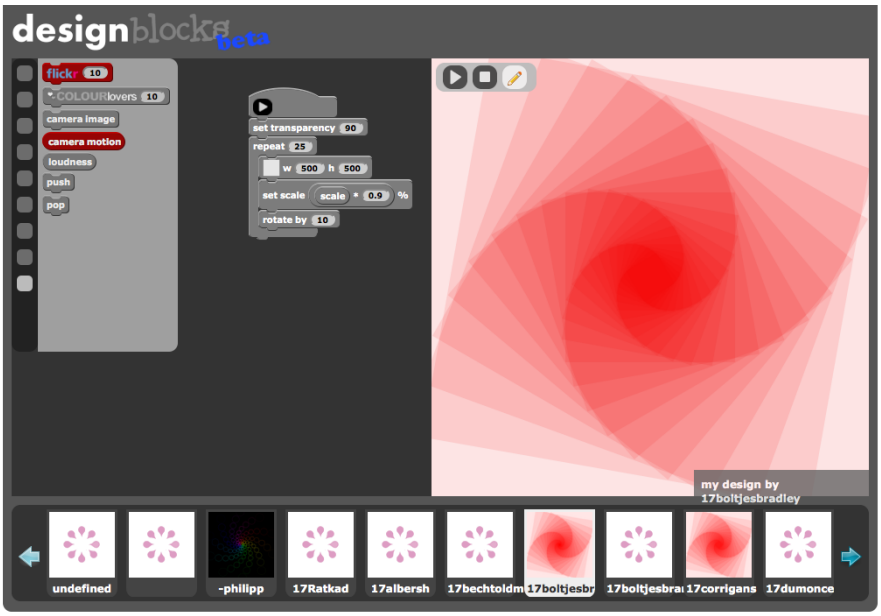



Figure 23: DesignBlocksJS

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 23) 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" (See 4.4 on page 27) for people that want to make their own modules but do not know web programming.

On touchscreens, coding with drag-and-drop codeblocks has the potential to be much easier than text-based coding. Some optimiza-

tion would be required to account for the difference in mouse pointer and finger size.

## 6.4 SOCKET COMMUNICATION

UX and server for sending arbitrary data from Meemoo on my smartphone to my laptop to your tablet (and back).

## 6.5 MEEMOO HARDWARE

Cheap computers (Raspberry Pi) + knobs + sliders + physical patch cables for performative interaction.

## 6.6 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
4. Collage that accepts images from many sources, like mobile phones
5. Text-to-song generator[1] 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 clips[2]
8. Hourglass module that flows virtual sand to other modules through the wires
9. Animated comic builder
10. Parallax-based 2.5D diorama builder
11. Arduino module to connect Meemoo apps to physical sensors
12. Two video players and a crossfader

1 Speech synthesizer in 1K of JavaScript by Mathieu 'p01' Henri: http://www.p01.org/releases/JS1K_Speech_Synthesizer/JS1K_Speech_Synthesizer.htm

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

13. (Animated) image macro generator
14. Side-scrolling game with placeholder image sprites that can be replaced
15. ...
16. TI-83 emulator[3] to draw frames of geometric animation
17. Logo emulator[4] that draws patterns with variables influenced by weather data
18. A Scratch game that draws different scenery based on location data

These examples show how – in the same way that the Internet encompasses 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.

3 TI-83 in JavaScript by Cemetech & Kerm Martian: http://www.cemetech.net/projects/jstified/jstified.php

4 Logo in JavaScript by Joshua Bell: http://www.calormen.com/Logo/

# 7 CONCLUSIONS

Call to action.

# 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 6.)

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 11.)

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 4.)

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 6.)

Himanen, Pekka. *The Hacker Ethic*. Random House Trade Paperbacks, 2001. (Cited on page 4.)

Jobs, Steve. "Introducing the First iPhone at Macworld." 2007. URL `http://youtu.be/6uW-E496FXg` (accessed on February 17, 2012) (Cited on page 4.)

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 6.)

———. "The Early History of Smalltalk." *History of programming languages II*. ACM, 1996, 511–598. (Cited on pages 6, 8, and 11.)

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 6.)

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 15.)

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. URL `http://74.125.93.132/search?q=cache:x5hrGaUFEZ8J:www.manovich.net/DOCS/manovich_new_media.doc+`

new+media+from+borges+to+html&cd=1&hl=en&ct=clnk&gl=us (Cited on page 6.)

———. *Software Takes Command*. Draft, 2008. URL http://lab.softwarestudies.com/2008/11/softbook.html (Cited on page 6.)

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 13.)

Morrison, J. Paul. *Flow-Based Programming: A New Approach to Application Development*. Createspace, 2010. URL http://books.google.com/books?id=R06TSQAACAAJ&pgis=1 (Cited on page 11.)

Morrow, Jennifer "Boriss". "User Experience talk at Mozilla WebFWD Summit." 2011. URL http://vid.ly/4o8d9l (Cited on pages 27 and 29.)

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 13.)

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 16.)

Papert, Seymour. *The Children's Machine: Rethinking School in the Age of the Computer*. Basic Books, 1993. (Cited on page 8.)

Papert, Seymour and Harel, Idit. "Situating Constructionism." *Constructionism*. Ablex, 1991. 1–11. (Cited on page 8.)

Papert, Seymour and Solomon, Cynthia. "Twenty Things to Do With a Computer." *Massachusetts Institute of Technology A. I. Laboratory* (1971). (Cited on page 35.)

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 3.)

Reas, Casey and Fry, Ben. *Processing: a programming handbook for visual designers and artists*. MIT Press, 2007. URL http://books.google.com/books?id=tqW75bfJkxIC&pgis=1 (Cited on pages 7 and 10.)

Resig, John. "Processing.js." 2008. (Cited on page 10.)

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 page 9.)

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 11.)

Shneiderman, Ben. "Direct Manipulation." *Proc IEEE Conference on Systems Man and Cybernetics* 97 (1986).December: 384–388. URL http://portal.acm.org/citation.cfm?doid=800276.810991 (Cited on page 26.)

Stallman, R. "The GNU manifesto." 1985. URL http://www.gnu.org/gnu/manifesto.html (accessed on February 5, 2012) (Cited on page 12.)

The Mozilla Foundation. "The Mozilla Manifesto." 2008. URL https://www.mozilla.org/about/manifesto.en.html (accessed on January 25, 2012) (Cited on page 25.)

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 7.)

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 13.)

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 16.)

Wichary, Marcin. "Guided Tour of Macintosh > Mousing around." 2005. URL http://www.guidebookgallery.org/tutorials/mac1984/mousingaround (accessed on February 10, 2012) (Cited on page 19.)

Williams, Sam. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly Media, Inc., 2012. (Cited on page 13.)

# APPENDIX

# A

# 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.

## A.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 description for the animation's page was a link to the Meemoo app for stop-motion animation 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.

## A.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 activates it and sees a 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 his desktop and then emails it to a friend.

### A.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.

# B

# CODE SAMPLES

## B.1 DEFINING INPUTS AND OUTPUTS (JAVASCRIPT)

```
Meemoo
  .setInfo({
    title: "example",
    author: "forresto",
    description: "this script defines 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"
    }
  });
```

```
{
  "info": {
    "title": "cam to gif",
    "author": "forresto",
    "description": "webcam to animated gif"
  },
  "nodes": [
    {
      "src": "http://forresto.github.com/meemoo-camcanvas/
          onionskin.html",
      "x": 128, "y": 45, "z": 0, "w": 343, "h": 280,
      "state": {
        "quality": 75,
        "width": 320,
        "height": 240
      },
      "id": 1
    },
    {
      "src": "http://forresto.github.com/meemoo-canvas2gif/canvas
          2gif.html",
      "x": 622, "y": 43, "z": 0, "w": 357, "h": 285,
      "state": {
        "delay": 200,
        "quality": 75
      },
      "id": 2
    }
  ],
  "edges": [
    {
      "source": [ 1, "image" ],
      "target": [ 2, "image" ]
    }
  ]
}
```