MEEMOO: CREATIVE HACKABLE WEB APPS

FORREST OLIPHANT

The main objective of this project is to design a modular dataflow visual programming framework using web technologies. The framework should empower non-coders to "hack" 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.

To this end, I have designed and created a web-based framework called Meemoo.

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 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 is programmed by connecting boxes with wires, 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 didn't 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.

The people that created the vision of the personal computer wanted everybody to be able to create their own tools. Learning traditional computer programming takes a major commitment, and most people are satisfied with the tools that come with the computer or are offered as services online. Meemoo is a toolmaker that makes it easier for people to create and modify their own tools.

In this thesis I describe...

ACKNOWLEDGEMENTS

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

CONTENTS

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 (Figure 1). 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. This interaction was part of an introductory program to teach mouse skills, called "Mousing Around."

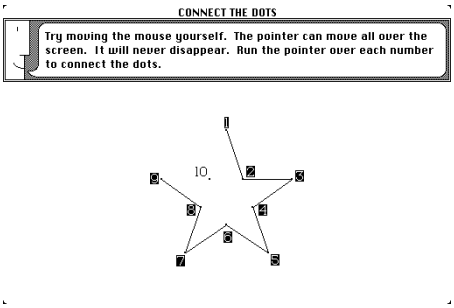


Figure 1: "Mousing Around"

Because of timing or school priorities, I wasn't part of the small generation of students that was exposed to BASIC or LOGO programming in elementary school. I remained interested in computers, spending any time that I could get my hands on them on shareware games and paint programs. I didn't 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 [=] the most times in one minute. 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 with these small machines. 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 wasn't discouraging. The serendipitous monochrome 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 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.

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.
> 2. One who programs enthusiastically (even obsessively) or who enjoys programming rather than just theorizing about programming.
> 3. A person capable of appreciating hack value.
> 4. A person who is good at programming quickly.
> 5. An expert at a particular program, or one who frequently does work using it or on it; as in 'a Unix hacker'. (Definitions 1 through 5 are correlated, and people who fit them congregate.)
> 6. An expert or enthusiast of any kind. One might be an astronomy hacker, for example.
> 7. One who enjoys the intellectual challenge of creatively overcoming or circumventing limitations.
> 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: adj. the ability to understand and modify the workings of a 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. Although it isn't 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 hacking.

Designing for hackability implies respect. The designer of a hackable thing acknowledges that they can't 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. While this enables other people with coding skills to modify a software project, I would like to expand the affordances of software hackability non-coders[3].

1 http://hackety.com/
2 http://hackasaurus.org/
3 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.

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 on the same tool. [1977]

> "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]

The term "Web 2.0" is used to describe 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. Web 2.0 makes media distribution easier by abstracting away the need to learn about web servers and HTML.

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.

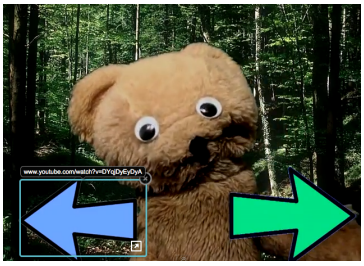


Figure 2: The Gilady Land Interactive Story

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."[4] One video is the entry into the story, and the rest of the videos are unlisted within the Youtube sytem. In the end of each section of the story, viewers are presented with two choices as hyperlinks within the video (Figure 2).

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"[5] 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 3).

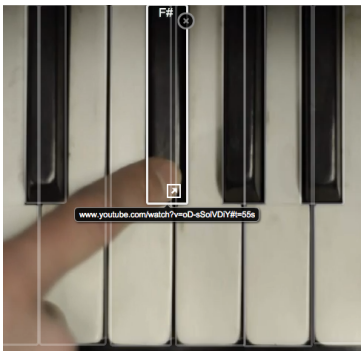


Figure 3: Play the Piano

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."[6] 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 4). 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.

4 "The Gilady Land Interactive Story" http://youtu.be/spVMyoUcuR4
5 "Play the piano" http://youtu.be/oD-sSolVDiY
6 "In B Flat" http://www.inbflat.net/

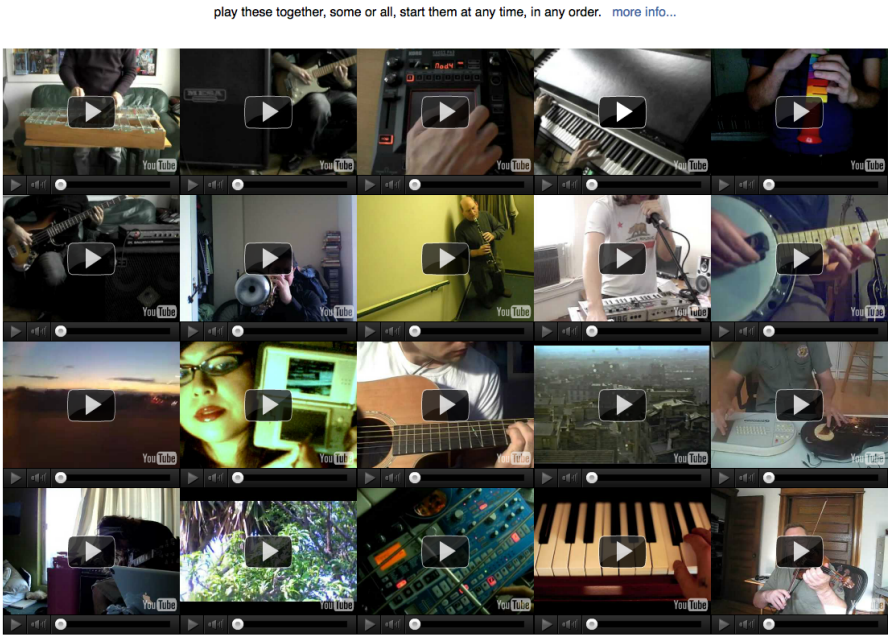


Figure 4: In B Flat

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

Youtube's embeddable player has a JavaScript Player API[7] which makes more complex interactivity possible. This increases Youtube's hackability, but again, only for people with coding skills.

1.3 *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]

The content is usually created offline, with digital tools like Photoshop that mimic traditional analog tools.

...

2 CONTEXT

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

2.1 *The personal computer*

The Dynabook was a research project of Xerox Palo Alto Research Center that envisioned and designed 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);

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

> 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]

2.2 *Programming for children*

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

2.2.1 *Logo*

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 this 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." He called this idea "constructionism." [Papert and Harel, 1991]

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].

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 5). 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].

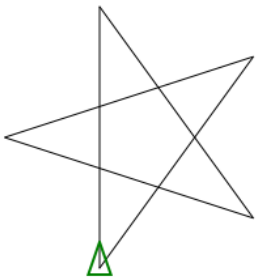


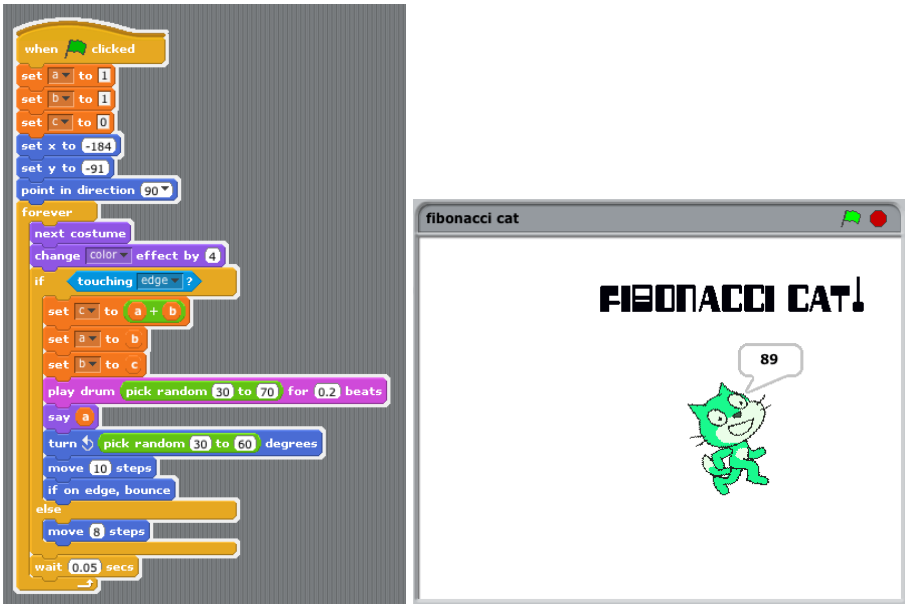
Figure 5: Logo turtle drawing a star

2.2.2 *Smalltalk*

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. Object-oriented programming splits code into logical classes that define the data for the object and the methods that access or modify that data. In tests some children even programmed their own tools, like a twelve-year-old girl's painting system and a fifteen-year-old boy's circuit design system. 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]

2.2.3 *Scratch*

Text-based programming languages have different requirements for syntax, punctuation, and indentation that if not followed perfectly will result in programs that don't run as expected (or at all). This can be frustrating for beginners and experienced coders alike—most of my programming errors are just missing semicolons. The creators of Scratch designed a system that takes the frustration of syntax errors out of coding.



(a) Script (b) Output

Figure 6: Fibonacci Cat! Scratch program by the author

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. "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]

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 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 6). I didn't 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'm an experienced coder, but I try new languages and coding systems on occasion, and Scratch was by far the fastest and easiest to get up to speed. I look forward to playing with it more.

One nice software design feature of Scratch is that 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 http://scratch.mit.edu/projects/forresto/2398409

2.3 *Visual programming languages*

Meemoo is a kind of dataflow visual programming environment. This means that on a programming

GRAIL (GRAphical Input Language) was an experimental dataflow environment developed by the Rand Corporation from 1967 to 1969. 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]

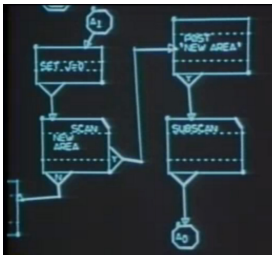


Figure 7: GRAIL

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]

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.

2.4 *Free Software movement*

Can't own ideas. Why do people give away their work?

2.5 *Open hardware and maker movement*

Arduino modules: Gameduino, heart rate sensor, Lilypad... abstracting some of the electronics intricacies into modular components.

"If you can't open it you don't own it."

2.6 *JQuery Plugins*

JQuery : plugin :: Meemoo : module

Large community sharing plugins. I made this thing and it is useful to me.

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.

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. They were able to do graphics and audio processing beyond my coding ability at the time, so I was able to explore new kinds of audio/visual experiments. They are great tools for interactive installations, but 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, reconfigurability, instant feedback, and shareability. Creating a new creative tool is just a matter of wiring some modules together. I can write new modules in code that I'm 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.

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

3.1 *Media Bitch (2002), Flash*

http://forresto.com/oldsite/interactive/mbx/mediabitch.html

3.2 *Kaleidocam (2007), Quartz Composer*

https://vimeo.com/387429

Learning QC and dataflow programming.

3.3 *Megacam (2010), Flash*

http://sembiki.com/megacam

Webcam apps inspired in part by Lomo cameras. I chose presets for each toy to make it simpler, but that also removed the possibility to experiment with the variables.

3.4 *Looplab (2010), Pure Data*

https://vimeo.com/16956269 http://www.flickr.com/photos/forresto/5125930908/

Learning Pure Data. Network communcation of identical apps, each passing data to the next.

3.5 *Opera stage 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, and projection mapping. I patched them together 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 one of them, and was able to do so in XCode.

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 and increase collaboration potential.

3.6 *Web Video Remixer (2011), HTML*

This is the direct parent project of Meemoo, where I figured out how to communicate between web pages in iframes.

4 DEVELOPMENT

Development on Meemoo's ancestor project began in January 2011. In October 2011 Meemoo became a Mozilla WebFWD fellow project.

Meemoo is designed for hackability on all levels. On the highest level, people can add and remove modules and reconfigure wires without coding knowledge. On the lowest level, the entire project is Free software under the MIT and AGPL licenses, which guarantee the right to fork the project and change how it works at any level.

4.1 *Software design for hackability*

One of the goals for the project is that it is hackable on all levels. On the lowest level, this means that the code is open source.

4.1.1 *Common communication library for modules*

Each Meemoo module needs to include meemoo.js, which handles message routing. The inputs and outputs are then specified as in Algorithm 1 on page 18.

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 position, connections, and state of all of the modules in the graph (Algorithm 2). 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."

4.3 *What is abstracted*

5 TESTS/RESULTS

5.1 *Persona*

One design research methodology that I used to think about potential audience for Meemoo was "persona" profiles. I defined three "persona" profiles to describe typical people at different levels of engagement with Meemoo: the creator, the hacker, and the modder. It is designed in a way that each of these levels leads to the next, encouraging people "down the rabbit hole" towards learning coding.

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.

Figure 8: Meemoo at Zodiak

5.1.1 *Creator*

5.1.2 *Hacker*

5.1.3 *Modder*

5.2 *User testing and feedback*

In order to test the user experience of the framework, I did in-person talk-aloud sessions. 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 aloud their thought process as much as possible.

Aino - Camdoodle
Ginger - "You should add an onionskin"
Teemu - Metronome animation
Jona - "Can I use this in my class?"
Facebook Beta group

5.3 *Economic model illustrated with Meemoo*

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

5.4 *Live animation visuals for dance party*

http://www.youtube.com/watch?v=T_tCyYGLWKM

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'm 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'm attracted to the textures and imperfections that come from using materials like these. Using the taptempo module, I

synced the sprites' animation to the rhythm. It was fun to build these tiny animations and then throw them onto the screens around the dance floor.

There are lots of improvements and ideas that came up in the evening:

- Camera: I used a Sony Eyetoy webcam, and the color was pretty bad. I chose kid's art supplies with rich colors, but most of the color was washed out in the first step. Next time I'll do some tests to find a better webcam, or use the camera on my phone, or a real digital camera somehow.
- 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. Directly un/replugging wires is a suggestion that is now high on the to-do list.
- 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'd like to take elements of 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 2) it will be relatively easy to make the community scalable.

6.2 *Touchscreen support*

Some media observers, myself included, saw the rise of touchscreen devices like the iPhone and iPad as a step backwards for participatory media. As originally marketed, these devices seemed to be designed primarily for media consumption. 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.

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

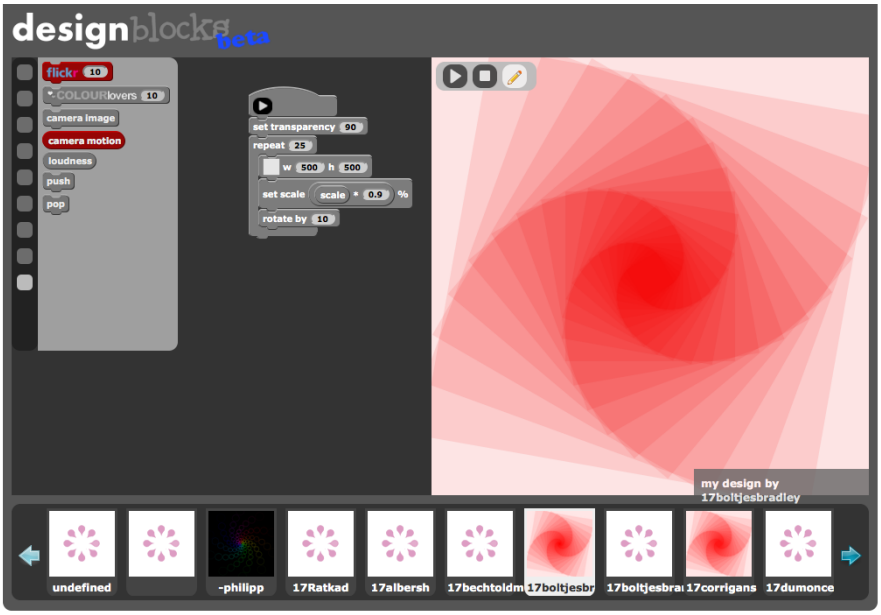


Figure 10: DesignBlocksJS

communicates, where things can be shared. The user then uses the app. The designer/user roles tend to be well-defined in this way.

The standard icon for an app looks like a shiny glass object (Figure 9), which mirrors the aesthetics of the device itself. It symbolizes something highly designed and polished, not to be opened.

Figure 9: App Icon

There are some notable exceptions: apps that encourage coding and exploration. These include Codea by Two Lives Left [9] for Lua coding, Processing.js Mini-IDE by Brian Jepson [10], and GLSL Studio by kode80 [11] 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 includes some well-designed features for touch-screen interaction 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.

Meemoo has the potential to become a powerful tool for creative programming on touchscreen devices. Gestures for zooming, panning, and dragging are common in touchscreen interaction, and should be tested to make them work with Meemoo. Zooming and panning already work smoothly, thanks to running in the browser.

There will be a library of modules that will reduce the need to write code.

Meemoo runs in browser, and JavaScript runs 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*

Scratch has inspired some open-source libraries that use a codeblock programming metaphor.

9 http://twolivesleft.com/Codea/
10 http://www.jepstone.net/blog/2010/04/16/processing-js-mini-ide-for-ipad-iphone-android-chrome/
11 http://glslstudio.com/

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 webcams pointed at screens
4. Text-to-song generator with computer generated voices singing in harmony
5. Artistic visualization of data from bio-sensors
6. Beatbox control of video mashup (sCrAmBlEd?HaCkZ!)
7. Hourglass module that flows virtual sand to other modules through the wires
8. TI-83 emulator [12] to draw animations
9. LOGO emulator [13] to draw animations
10. A Scratch game that draws different scenery based on location, time, and weather data.
11. ...

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.

7 CONCLUSIONS

I contacted Ze Frank to ask if he would be a project advisor. He gave me some good things to think about:

> "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, etc..."

Making a creative tool maker is pointless if, in the end, nothing creative is made. My dream is that somebody will make something beautiful with it. Shouldn't that somebody be me? If I don't do it, why would anybody else?

12 Proof-of-concept by Cemetech & Kerm Martian: http://www.cemetech.net/projects/jstified/jstified.php
13 Proof-of-concept by Joshua Bell: http://www.calormen.com/Logo/

T. O. Ellis, J. F. Heafner, and W. L. Sibley. The GRAIL language and operations. 1969. URL http://www.rand.org/pubs/research_memoranda/RM6001.html?RM-6001.

Alan Kay. User Interface: A Personal View. In Brenda Laurel, editor, *The Art of HumanComputer Interface Design*, pages 191–207. Addison-Wesley, 1990. ISBN 0201517973.

Alan Kay. The Early History of Smalltalk. In *History of programming languages II*, pages 511–598. ACM, 1996. ISBN 0897915712. URL http://dl.acm.org/citation.cfm?id=1057828.

Alan Kay and Adele Goldberg. Personal Dynamic Media. *Computer*, 10(3):31–41, 1977. ISSN 00189162. doi: 10.1109/C-M.1977.217672. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1646405.

Lev Manovich. *Software Takes Command*. Draft, 2008. URL http://lab.softwarestudies.com/2008/11/softbook.html.

J. Paul Morrison. *Flow-Based Programming: A New Approach to Application Development*. Createspace, 2010. ISBN 1451542321. URL http://books.google.com/books?id=R06TSQAACAAJ&pgis=1.

Seymour Papert. *The Children's Machine: Rethinking School in the Age of the Computer*. Basic Books, 1993. ISBN 0465018300. URL http://books.google.com/books?hl=en&lr=&id=q9x7sx901KwC&oi=fnd&pg=PR7&dq=The+Childrens+machine&ots=42cAth2UOs&sig=DV6M6-wGW37G7b-1yj-N9aVxo2g.

Seymour Papert and Idit Harel. *Situating constructionism*. 1991. URL http://namodemello.com.br/pdf/tendencias/situatingconstrutivism.pdf.

Eric S. Raymond. The Online Jargon File, version 4.4.8: Hacker, 2003. URL http://catb.org/jargon/html/H/hacker.html.

Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. Scratch: Programming for All. *Communications of the ACM*, 52(11):60–67, 2009. ISSN 00010782. doi: 10.1145/1592761.1592779. URL http://portal.acm.org/citation.cfm?doid=1592761.1592779.

Ben Shneiderman. Direct Manipulation. *Proc IEEE Conference on Systems Man and Cybernetics*, 97(December):384–388, 1986. doi: 10.1145/800276.810991. URL http://portal.acm.org/citation.cfm?doid=800276.810991.

Algorithm 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"
    }
  });
```

Algorithm 2 Meemoo App Source Code (JSON)

```
{
  "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/canvas2gif.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" ]
    }
  ]
}
```