MEEMOO: HACKABLE CREATIVE WEB APPS

FORREST OLIPHANT

The main objective of this project is to design a modular flow-based visual programming environment using web technologies. The environment should empower non-coders to "hack" creative web apps by configuring wires that represent how modules communicate. Apps created with the environment should have source code that is easy to read and share.

I have named the framework Meemoo. Within this framework, an app is a graph 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 graph that defines an app's layout, routing, and state can be saved and shared with a small amount of text.

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 data-flow graph can be made into a Meemoo app.

Meemoo has been designed with a few groups of people in mind: creators, hackers, and modders. 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.

In this thesis I describe...

CONTENTS

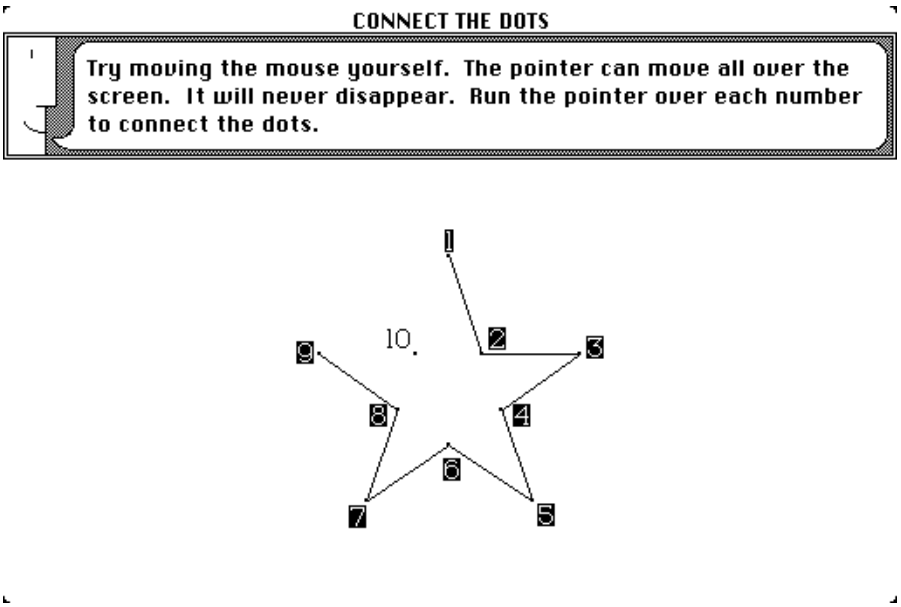

Figure 1: "Mousing Around" [Wichary, 2005]

1 INTRODUCTION

My first memory of interacting with a personal 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). This interaction was part of an introductory program to teach mouse skills, called "Mousing Around." Seeing this graphic, however simple, react to my input captured my imagination. We only had that computer for a few days, but I was hooked.

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: the Texas Instruments graphing calculators and web programming.

My higher-level math classes used TI-83 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 figured out how to script complicated graphic drawings with these small machines. It was satisfying to see the calculator slowly render geometric patterns from my scripts. This was my first experience with programming graphics. I never managed to make the 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 believe that I learned a lot about geometry, algebra, and logic from these code explorations, but not in a way that was applicable to math tests.

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 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.

1.1 *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.
> 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.

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 hack, bend, mod, or fork it to their will, and connect it to other things. This quality can apply to software, electronics, and other

1.2 *Metamedia*

Kay: computer can be all other media, and is also active. Internet makes all media available everywhere. Web 2.0 made it participatory for publishing text, photos, videos. Twitter, Flickr, Youtube: one solution per media, no coding needed.

Web 2.0 makes media distribution easier by abstracting away FTP, HTML, etc.

Maybe Web 2.5 is creative apps online, like Picnik photo editor and TinkerCAD 3D design software. Media creation without desktop software.

Meemoo makes the modes of production also participatory. This makes it possible for people to invent and share new media. Web 3.0: web software creation without desktop software (or code)?

If, as McLuhan proposed, "the medium is the message," then what kinds of messages are implied by the metamedium?

D. Rushkoff argues (Program or Be Programmed) that learning to code opens people's eyes to the design of all systems. They see that bad design can be fixed.

1.3 *Tools*

From making tools to making toolmakers. Tool designer thinks: "How will people use this tool?" Toolmaker designer thinks: how will tool designers use this tool (to think about how people will use their tool)? Thinking about thinking about thinking (about thinking?). Metaconstructionism?

Meemoo will enable people to become tool designers before learning coding skills.

Programming skills distinct from coding skills. Visual programming makes the dataflow logic visual. With text-based coding you need to keep track of these relationships in another way.

2 CONTEXT

2.1 *Computers as abstraction*

2.1.1 *Mainframe to PC: Dynabook*

Kay: Promethian effort to bring computation from "priesthood" of mainframe admins to all people, especially children.

2.1.2 *Programming for kids: Smalltalk, Logo, Scratch, Alice*

Constructionism: getting kids programming gets them to think about thinking. The metamedium gives them control.

2.2 *Visual programming languages*

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

2.3 *Free Software movement*

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

2.4 *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."

2.5 *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

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

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, experiement, 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.

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 12.

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 cord-type manually operated telephone switchboards (if not even earlier, possibly for telegraph circuits). Cords with plugs at both ends had been used for many decades before the advent of Dr. Moog's synthesizers to make temporary connections ("patches") in such places as radio and recording studios. These came to be known as "patch cords", and that term was also used for Moog modular systems. As familiarity developed, a given setup of the synthesizer (both cord connections and knob settings) came to be referred to as a "patch", and the term has persisted, applying to systems that do not use patch cords." - Wikipedia on Moog

4.3 *What is abstracted*

As a programmer, working in Puredata and Quartz Composer can be frustrating. Certain logical constructions that would be easy to describe in code become a jumbled mess of boxes and wires.

Figure 2: Meemoo at Zodiak

5 TESTS/RESULTS

5.1 *User testing and feedback*

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

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*

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. Because of the small amount of source code to describe a Meemoo app (Algorithm 2) it will be easy to build a scalable platform for sharing and forking apps.

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 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 3), which mirrors the aesthetics of the device itself. It symbolizes something highly designed and polished, not to be opened.

There are some notable exceptions: apps that encourage coding and exploration. These include Codea by Two Lives Left [1] for Lua coding, Processing.js Mini-IDE by Brian Jepson [2], and GLSL Studio by kode80 [3] 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.

Figure 3: App Icon

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 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 *Socket communication*

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

6.4 *Meemoo hardware*

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

6.5 *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 [4] to draw animations
9. LOGO emulator [5] to draw animations

1 http://twolivesleft.com/Codea/
2 http://www.jepstone.net/blog/2010/04/16/processing-js-mini-ide-for-ipad-iphone-android-chrome/
3 http://glslstudio.com/
4 Proof-of-concept by Cemetech & Kerm Martian: http://www.cemetech.net/projects/jstified/jstified.php
5 Proof-of-concept by Joshua Bell: http://www.calormen.com/Logo/

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?

REFERENCES

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

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.

Marcin Wichary. Guided Tour of Macintosh > Mousing around, 2005. URL http://www.guidebookgallery.org/tutorials/mac1984/mousingaround.

APPENDIX

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" ]
    }
  ]
}
```