Ontario College of Art and Design
University

screenPerfect: the importance of accessible technology for artistic use.

Author:
Alex Leitch

Supervisor:
Dr. Emma Westecott

A thesis submitted in fulfilment of the requirements
for the degree of Masters of Design

in

Digital Futures

in the Faculty of Design

11th February 2014

Declaration of Authorship

I, Alex Leitch, declare that this thesis titled, 'screenPerfect: the importance of accessible technology for artistic use.' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a master's degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

I wish to express my appreciation of my thesis advisors, Emma Westecott and Simone Jones, without whose generous contribution of time and sensible advice this would be a much weaker paper. I would also like to express gratitude to the Site 3 coLaboratory community, who provided me space to work and a community to work with, and Bento Miso - Dann Toliver, Cecily Carver, Jennie and Henry Faber expressly - for their technical advice and their help in hosting No Jam 2. I would also like to extend my thanks to Evan of the Digital Futures tech desk for a key tip when I was just starting my research. Hannah Epstein deserves my thanks as well for being a fantastic collaborator.

To my personal support community, many thanks for your patience with me for the duration of this work.

For Adina....

Contents

Abbreviations

CYOA **C**hoose **Y**our **O**wn **A**dventure
A popular format for game narrative, describing a branched, choice-based structure, as versus a linear story.

FiG **F**eminists **i**n **G**ames
A SSHRC-funded association of digital researchers interested in disrupting gender bias in game development.

DMG **D**ames **M**aking **G**ames
A non-profit community organization based in Toronto dedicated to supporting dames interested in making, playing, and changing games.

JS **J**ava**S**cript
A scripting language, traditionally used for client-side programming on the web.

Indie **I**ndependent Developer
Indie developers are game developers not associated with tradtional, well-funded development houses.

1 Introduction: A Better Time-Based Installation

1.1 Introduction

What constitutes good design? As expressed by Don Norman in the book "The Design of Everyday Things," design can be sympathetic to their users, or have psychopathy coded directly into their construction. ScreenPerfect has been designed to address the question of how we might improve software based on internet technologies to be more useful to artists.

This project has several parts, which will be explained separately. The first part of the project is an application to build branched narratives out of video files, coded in NodeJS for distribution via common internet technologies. This portion of the research has to do with the idea of resistance and how people's consciousness and abilities can be expanded by using contemporary technology, which is made by trained technicians but then released to be used without a proscriptive definition of the use case. Users make use of the engine, developed through one specific user's design practice, to design new things. The first part addresses the importance of this type of engine to generation of contemporary art experiences.

The second part of this paper addresses how to perform user testing via a game jam format, a type of design charette in which users are given access to tools and asked to produce art with them. In that section, I document how No Jam 2: videovideo worked, the industry paired advantages to new engine development, and how this expands community ties for local artists working in a medium that frequently demands more collaboration than traditional, less dynamic art works.

In the third part of this paper, I address the issue of new media art and display. New media, particularly time-based media, is very challenging to display and support. Even low-end computers are bulky and expensive to deploy, so I have asked the question

of how we might circumvent traditional white-cube galleries while restricted by the requirements of internet technologies. In this part I detail how to build a server on the raspberry pi platform, a microcomputer, to deploy web-based applications to localized, internet-restricted spaces.

In conclusion, I argue that we need technology to belong to its users, instead of pursuing an exclusive reliance on mass network technologies. I feel that there has always been room for the technical in art, and while good technology fades into the background to leave only the artwork on display, the technology we choose to use gives us a frame of what can be pursued.

1.2 Designing Software to Power Experience

The arts and the humanities are the technical name for the fields of work that produce both North America's culture and its record of its culture. We use computers to do the work, because they extend our ability to speak in repeatable patterns. Repetition is a key component of mass production. Looked at as a tool, a computer is not so much a hammer as it is an ongoing negotiation - the user must decide what the black box means Glanville, 2014.

The use of computers as tools is a specific skill set, which is as unique as the skill set of using a paintbrush. The key element of computer skills is a comfort with curiosity: digital tools change all the time, and many of them are not well written. Software is frequently unreliable, and hardware moreso. Almost all software tools require a vast period of time invested in skill acquisition before content - art - can be produced, and this time is expensive. The construction of the tools themselves is an art, because when manufacturing a tool, it needs to be easily used, but it also needs to do something in a predictable, reliable way. A good digital tool should encourage rather than impede expression.

Artists, as a rule, have their own working methods and vision for their work in advance of picking up any new tool. For the purposes of this work, it is assumed that users have their own vision independent of the tool itself, which can be expressed, expanded, or extended by the use of a new tool. Ideally, a tool vanishes in its use. It is a multiplier of force, where force can also be capital, which can be either an intellectual weight - the gravity of a cultural construct distending other expressions - or the more conventional "money."

Television and video are a format requiring intense capital to access. Culturally, video works - film, television - express mass ideals to mass audiences. YouTube, Vimeo,

the late Steve Job's well-documented interest in the format, and visual FX production continue to drive technological innovation in systems development. Most movies have some vFX to them, a capital-intense practice that requires hours of work. Acquiring editing skills and image-design skills to work with moving pictures is an expensive and time-consuming pursuit, which has nonetheless become more open in recent years. The advent of Vine and YouTube has democratized video to the point where it can be traded as words once were.

This does not make the expression of capital inherent to these works more valuable, but less. It becomes apparently disposable, consumable on devices that are as close to us as clothing. Video is consumable on every kind of screen, especially on the smartphone screen. On one level, this is great for audience acquisition. On another level, it shifts the import of the work from the work to the container. Though the medium has been the message, in the contemporary internet, the envelope is the note.

Fortunately, in the contemporary internet, the envelope is also the entire postal service. A single smartphone in 2014 is more than powerful enough to supply most of the switching and serving needs of previous video works, including branched narratives that once required many VCRs and multiple screens. As things become more accessible, they lose their Walter Benjamin aura, their singular magic, which means that accessing that magic becomes more challenging as the distribution of the work becomes easier. Partly, this seems to be a process of decontextualization: no matter how good a given film, it may be better in a theatre, en masse, because the theatre, the production of context and attendance, makes the experience of the film singular, even as the film itself is limitlessly reproducible.

So the question becomes: how to best retain the capital of artistic tool use, vision, and creative practice, while restoring the aura of presentation required to engage with art on its own terms? How to expand artistic tool use into new means of expression? How to make use of contemporary methods in a way that may remain accessible and on display for years to come?

1.3 Interaction and Presentation in Game Design

Games, particularly the subset of video games known as triple-A, are expensive to make and practically require a team of people to produce. This has been seen as restricting the degree to which the stories these experiences communicate can be personalized. While there are counter-examples, such as 2013's excellent "Saint's Row 4," many triple-A games need to be able to make back their budget, which restricts their intended audience

to those who can pay to play. The games themselves utilize engines which are generally private or closed-source. An engine is the software that dictates how the physics and scripting that contains a game world can be manipulated. Many are closed-source and privatized, and therefore expensive. All of the engines as they presently exist require not only the skill of framing and lighting an engaging experience, but also a panoply of other skills - character modelling, animation, colour theory, programming or scripting, and sound.

This creates a design challenge: How to best reduce the barriers to entry in game-making to encourage new voices?

A new type of game engine is one possible approach. An engine is the body of software that drives all interactions in-game. Assets, such as artwork, music, animations, and scripts to dictate how these assets are integrated, are all added to an engine that provides a framework to drive any given game. Some engines encourage more experimentation than others. The Twine engine, for example, is designed to provide branched, highly-stylized text narrative to a web browser, and it has been adopted by its userbase to tell detailed stories that are highly personal - the sort of work that cannot always be addressed by games with a bigger budget. Twines are limited in form but not in scope. The Unity engine provides traditional assets and scripting, and has been used by independent developers to produce games such as Gone Home (2013), a work about a missing family mainly told through examining objects and listening to music. Twine takes advantage of an author's skill at pacing and writing to divide a narrative into a choose-your-own-adventure work, paced through timed links and designed to take advantage of the detailed design possibilities of text in the browser.

Independent games are typically distributed through the internet, or rely on the internet for their entire lifecycle. This is problematic, not only because the shape of the engine dictates the shape of the experiences that can be produced. The internet is owned, mainly, by very few extremely wealthy people, and the technology is fragile. There are many ways to lose access, from legal means such as France's HADOPI laws, by which whole households can have their access cut off, to a simple lack of bandwidth in an installation space. This causes huge problems for both exhibitions and archives, as art based on access to the external web can vanish with no warning. This is also unacceptable for institutional collection. The availability of web art combined with its unreliability devalues the work of the artists who have created it. Art that relies on the network is simultaneously omnipresent and vanishing, capable of accessing a mass audience and disappearing at the moment when a local audience is available.

Audience definition becomes important in this context. A browser-based or internet-distributed game has the possibility of reaching a very broad audience - millions of

users. The artist has no guarantee of the context of their work in the view of the audience, beyond that it is likely to be screen-based, viewed on a personal or work machine. Perhaps this works. Whether or not screen-based art as it presently exists is effective is outside the scope of this paper, however. Within the scope of this work is that digital work is difficult to display *outside* the context of this mass market.

If electronic art is to be included in large collections, or displayed privately, or reproduced such that it benefits mainly the artist rather than the distributor, there needs to be a means to display it that does not rely on external resource providers. This includes the easily-considered difficulty of network providers as well as the more challenging to contextualize power grid. Both are unreliable in the permanent sense, where the relatively small amounts of power and information (which are sometimes the same things) can actually be supplied locally.

The localization of a broad audience - how to supply a thousand or ten thousand people simultaneously with a single experience - is outside the scope of this paper. This design work instead addresses the question of how to bring a work built for broad distribution into a narrow context for better engagement via a system of resilient display. This system uses local resources rather than relying on global supply to always be there.

This touches on themes of paranoia, privacy, and the resistance techniques beloved of both cybernetics and French post-structural feminism. There is a sense of play in code: there are many, many ways to achieve a topically identical personal experience using software.

Web technology is important to software development for a variety of reasons, part of which is that it is already more accessible to novice users than traditional, desktop compiled software. Web browsers have offered users the ability to view the source code of the pages that users are reading, which includes the content of scripts. With the advent of developer tools included standard in browsers, it has become straightforward to build and test software ideas quickly in an environment provided on every operating system. This is now under threat with the advance of embedded digital rights management software **arstechnicadrm** but it remains an important learning resource, because it is available with every desktop computing platform.

People who have little time cannot afford to acquire the skills to use a new and complex tool. Due to these restrictions, the number of people for whom computer use in and of itself will be a delight is a limited population. This is problematic, because art production is difficult and time-consuming even without the boundaries raised by software challenges. The more limited and specific the skill set required to use contemporary

software tools, the more difficult it is to include a diversity of voices in the cultural production of genuinely contemporary work. When artists are excluded from technology, culture splits on lines of privilege. There are artists who make art, and technologists, who make technology, but do not see themselves as particularly responsible for the ideas encoded in their work.

Technology is not neutral. It is authored, and where there is authorship, there is a responsibility for ideas. When large groups are left out of communication media, particularly those tasked with producing the language with which culture speaks to itself, there comes a disconnect in the public representation of our sense of self.

The problem of interesting experiences is not trivial. Video games offer an economically advantageous distraction engine, a way to enact an artificial life during a period of declining general wealth. Allowing a diverse range of voices easy access to portray their own games, their own alternate or idealized modes of being, is a way of making those voices more real, of offering an alternate human experience to the "asshole simulator" (Bissell, 2013) genres manufactured at much higher budgets.

1.4 Initial Approach

1.4.1 Federal Development Grant, game::play Lab, and collaborative artistic practices

The initial code of screenPerfect came about as part of a collaborative development project in OCADu's game::play lab to produce a vision of how dual-screen artworks might work going forward. The original software powered a game called psXXYborg, produced by Hannah Epstein under the supervision of Emma Westecott. From there, I became curious as to how we could transform the engine software to include game-editing tools, to encourage a wider range of video artists to use the software. This became the basis of the initial portion of my thesis work, the screenPerfect engine. In order to generate sufficient games to demonstrate the software, and to figure out where the software could be improved, we then partnered with Bento Miso coworking space in a mutually beneficial game jam called No Jam 2.

No Jam 2 featured both an editing segment and a re-architected version of screenPerfect that uses Bento's new language, Daimio, designed mainly for open use on the internet. After the jam, the games were collected, with their resources, and screenPerfect was forked to become two separate engines. The original engine was retained for displaying works to that point, and a new engine called iV to promote ease of access within the

Dames Making Games, a feminist social group run at Miso by the same developers who worked on the engine.

iV differs from screenPerfect in that it drops the dual-screen element and encourages a single-focus narrative. ScreeenPerfect has also been reoptimized for local display of static files from a single Linux-based computer. I selected a raspberry Pi, both because of its scale and affordability, and because it stems from a collaborative learning foundation in the UK.

There is no part of this work which is not an assembly of other works, which does not rely on the network of the internet being at least in part an open place to learn how to build software. This is an essentially agile, iterative development process, wherein the first type of software was delivered, and everything that has emerged afterwards has been an evolution of the initial design concepts, which were themselves an end goal. This is programming from back to front - hacking, in Sadie Plant's definition, a practice where one starts at the end of what one is looking for and then sews up every step on the way until the software works as intended Plant, 1997. This type of systems design is holistic, borrowing heavily from independent game design and general creative practice.

The formal agile method of software development is based on the idea of delivering working code in advance of documentation, and putting the user ahead of the planner in software design. I have included a summary of the method in Appendix A, The Agile Manifesto, but what it chiefly means is less advance software planning of every detail, and more asking questions about the best way to solve a problem in code.

This thesis assumes that the definition of gaming as an art form dependent on action (Galloway, 2006) is accurate, and therefore, after developing an initial round of software, the main body of feedback and development in this software package has been via user collaboration. The volunteers can then give feedback to both the software developers and to each other on what is possible with the tool. At that point, I will incorporate their feedback into future versions of the tool, hopefully leading to a stable platform for interactive experience development. This is a dynamic type of user-centered design, relying heavily on version tracking software and rapid updates, in line with agile development ideals.

The initial software underlying screenPerfect has been developed in concert with an artist who laid out an idea for how a video interaction might work, which has then been created and refined, released to more artists, and then revised again. The hope is that each new version of the tool will generate a useful echo chamber, amplifying new ideas even as it makes advanced technology easily accessible to content producers.

The toolset can then be released and left for artists to use and analyse, and can be expected to run privately on optimized systems similar to "game cartridges" from the 1990s.

FIGURE 1.1: Game Cartridge of Chrono Trigger, 1995

1.4.2 Code and Theory

The initial software of this project was developed as a response to the lack of privacy and control of various shared media sources online. Rather than developing for a mass audience, I began with the idea that this should be developed privately, for small audiences using disconnected technology. Over the course of many installations of the psXXYborg software package, I noticed many installation issues, many dependent on a reliance on external resources.

The code of screenPerfect was written in Javascript on the server and client side, using Node.JS. Node is a server environment library and framework that is intended to permit web developers familiar with JS to write their code to the server. During the process of the thesis, the idea of screenPerfect as a game engine was taken up by my industry partners, Bento Miso, who are interested in the idea of new game engines as a use case for their language, Daimio (Faber, 2013). In the case of ScreenPerfect, I decided to write the initial application in Node.JS and javascript to take advantage of the speed of the Google V8 code engine. When that application was done, I turned it over to Miso, who refactored the code into something that could be attached to the Daimio language system.

The critical theory that underlies this practice is a combination of French poststructuralism - Helene Cixous in particular - and contemporary writing on video games and the history of women in technology. By producing the software and content with the input of a local feminist collective, Dames Making Games, I have grounded the work in a social justice driven practice which encourages women to take part in their own lives by learning how to interact with machines and communicate with the broader world.

1.4.3 Game Engine Research

The Twine engine is a branching narrative engine for authoring text narratives. I have taken the idea of Twine and transformed it to use video and still images to expand the possibilities of an author experience.Twine is affordable and requires very little training

to access, but it is also limiting, in that it undermines skillsets in imagemaking which may already be highly realized.

Video games, particularly console-based video games, have a huge leg up on experimental art in one basic respect. Video games are still generally expected to run when the internet vanishes. The Nintendo 3DS, a pocket console, outsold every other system on the market in 2013 **nintendosales** It is speculated that its success is due largely to the fact that the 3DS is a portable system that does not connect to the broader internet unsupervised. You purchase games for it on cartridges, which contain that piece of software and no others. This makes the system portable, unlikely to be co-opted by malware, and controllable, in that it is tough to get to the wilder areas of the internet unsupervised. This reliability is something that is rare to find in more complex computers: sometimes, as argued by Don Norman in "The Design of Everyday Things" Norman, 2002, it is best to have a single thing do one thing really well.

Cartridge design is excellent, and limited. Cartridges historically fit into one system, once, and rely on details of that system's hardware to run. They still run on their original hardware as much or more than twenty years later, provided the original hardware can be kept to spec, where newer software, reliant on repeated patching after launch, may fail more frequently.

The other good example of portable electronic interaction is the smartphone. Mobile is a huge segment of the market, which is excellent for text messaging, talking, and playing games that separate an audience from each other, but not so great for bringing people together in the same space. Part of the appeal of the smartphone is that it is customizable, which means that each user can design their own experience and use it.

There are games that choose to subvert this separation, and systems have been built to take advantage of the power of pocket computers. The most notable effort to date is "Space Team," a simon-says game for teams of up to four players. The application pairs to itself across phones by using a common network connection, and players in the same physical space cooperate to pilot a star ship. This is excellent, as it allows players to make use of a device with which they are already comfortable to cooperate and share an experience.

Unfortunately, it is limited to Apple users, because SpaceTeam is an application, limited to a single system.

The idea underlying screenPerfect is that all users have access to the internet on their pocket devices, and the internet is both bigger and smaller than the network which delivers content to users. Rather than relying on The Internet, screenPerfect provides a private wiFi point and what is called a "captive portal" to let players pair with one

another and the server, control a large screen, and interact with a piece of video art in a localized area. This means that an artist can control the exhibition space for their work, design the experience of the work, and ensure that their audience will experience the work in a context that makes sense. It aso ensures that technicians can access the underlying engine should something go wrong during the installation.

1.4.4 Experiments in System Control within Cybernetics and Feminism

This work is related to various texts of feminist or woman-oriented critical theory that have appeared in the years since: Haraway's Cyborg Manifesto, TIQQUN's Preliminary Materials Towards A Theory of the Young-Girl. These are academic constructions of femininity as it is seen in relation to technology: they are feminist in the formal sense of the word. There are other senses of the term, which I will not be examining within the paper. Rather than expressing this work in context with Cixous as *écriture feminine*, I will be using Cixous as a reference for the idea of the alien perspective as a position of resistance within a means of expression controlled by a neutral-to-hostile majority.

This approach addresses women as an alien construct to the more conventional world of technology, which has been recently associated with a masculinist performance that is unnecessary for the pure structure of good rules and the development, through that, of good software. This construction is relatively recent, as Nathan Ensmenger presents within his work on the systematic exclusion of women from programming as a trade (Ensmenger, 2010).

Accessibility for artists is a major contemporary concern, related to the development of languages like Processing by Ben Fry and Casey Reas (**processing**), microcontrollers like the Arduino platform by Massimo Banzi co. (**arduino**), and frameworks like Scratch from the MIT Lifelong Kindergarten Group (**scratch**).

People who write code to develop an idea are engaging in a conceptual creative practice themselves, and they then display that creative practice through the artists who use their work. Purposefully producing small tools to support creative practice is a different model than that of major software development houses, where the work is designed in isolation, iterated on a strict schedule - 18 months in the case of Adobe **adobe** - and released. There are documented design methods for software that follows this model - I have listed Agile as the most well-known - but for the most part, this is simply considered an interative working practice which aims to a functional, finished result.

Although artists are the central agents of production of the invisible yet tangible value of the culture industry, they are not the prime beneficiaries of the financial system that backs, stores, and distributes the results of that capital. This is capital as both skill and capital as resource distribution: computers are expensive. Getting around that face is important if we wish to supply access to contemporary media to a broad array of voices: to be inclusive, software should be the last thing that gets in the way of a work, rather than the first. This reserves the value of scarcity - the market value of the work - to the *ability* of the artist, rather than applying the majority value to the role of the engineer. This kind of invisibility is the invisibility of good management, of any type of good administration. Like housekeeping, code recedes until something goes wrong.

To test this idea, I have approached people to produce video—games with the screenPerfect software in the context of a voluntary game jam – a type of collaborative space where participants work with digital tools to generate new, raw games in a limited window – and then compiling the results into an arcade machine for presentation.

2 Background, Theory, and the State of the Art

2.1 Existing Software

ScreenPerfect does not exist in a void. Although written fresh in Javascript, it is dependent on many frameworks and libraries in order to work. The code was developed using the Node framework for Javascript on the server, which is supported by Google. It mimics functionality produced by the Dataton Watchout system, which provides simultaneous video windows using a custom, private hardware platform. The software that screenPerfect interacts best with is Google Chrome.

2.1.1 Game Engines: What Are They?

A game engine is a collection of software designed to make it possible for a team of artists, developers, musicians, and producers to work together to produce a complete product. Traditionally, game engines are used to produce 2D or 3D experiences with clear "assets" such as 2D sprites or 3D player character/interaction models, backgrounds, interaction assets - crates, for example - music, and scripts in a programming language to tie all of these together into a play experience.

Some popular professional engines at the time of writing are Unity3D, which features native mobile integration and ease of scripting in both Javascript and C, Crytek, which comes with many high-end 3D resources preloaded for high definition graphic support, the Unreal Engine, which is quite stable and useful to experienced teams that prefer more control over their work.

There are popular hobby engines that de-emphasise programming as well, such as GameMaker, which is prized for PC compatibility, Game Salad for OSX, and Construct 2, which is PC-only but has a powerful engine to manage game physics and interactions.

These engines all assume a certain type of player interaction: they are designed to enable designers to produce specific types of games, such as a "shooter" or a "platformer", similar in style to the Call of Duty franchise or Nintendo's Mario series. The interactions available are easily understood as a language of action by their players, provided players have previous experience with video game play.

ScreenPerfect is distinct from pre-existing engines. It is a piece of software custom written to encourage artists to use their own skillset in image and video creation to explore what is possible in an interactive experience. Screenperfect has a set of play mechanics that have been pre-written. While they can be *extended* by anyone who knows the Faber, 2013 language, the mechanics are straightforward to use and not designed to be altered by artists. This means that artists have a consistent environment in which to place their work, which will reliably showcase that work without them needing to learn how to program - an entirely new creative skillset - to do so.

2.1.2 Twine

Twine is the closest game engine to screenPerfect at the time of writing. Twine is a locally-installed hypertext-based branching narrative platform, which produces an interactive narrative that can be accessed through any web browser. It encourages expressive type styling and elements of multimedia, including music, coloured type, and well-designed game screens, but does not require them. Twine does not yet support video narratives, and is not entirely stored online as of yet.

The Twine engine was popularized by indie gaming celebrity Anna Anthropy in her 2012 book Rise of the Videogame Zinesters Anthropy, 2012, p. 2012. Since then, hundreds of Twines have gone live.

Twine is most notably popular with the queer indie gaming community. It has been used in wide popular release by Anna Anthropy, Merrit Kopas - author of *Lim*, Porpentine, Zöe Quinn, and games critic Soha El-Sabaawi, among others.

2.2 Multiscreen Video Technology

Dual screen technology, or more accurately, multi-screen synchronous web technology, is one of the big new ideas being heavily backed by Google in 2013. As a consequence, its Chrome browser has been designed to support software developed with a specific suite of frameworks, many of which are wholly supported by Google. This is an example of how software is not free: we cannot guarantee *what* is being done with the whole of

our software installations, or whether there is a security flaw in a system written to be dependent on development tools from major software houses.

That being said, Google supports Node and Chrome both, so multi-screen technology using web browsers is accessible to people for no more investment than a new language, for the moment. Google, like many future-facing organizations, has a bad history of dismissing old technologies for the benefit of the new, but Chrome seems stable enough for now. ScreenPerfect relies on Node.JS, which is based on Google's V8 engine, and MongoDB for databasing. In addition, although the engine was originally authored independently using exclusively Javascript, later versions have been reauthored using the Faber, 2013 dataflow language to describe connections between game files. Daimio has been released under the MIT licence by Bento Miso in Toronto.

The architecture of screenPerfect is wholly new, but the concept is based on the Dataton Watchout system, which encourages producers to develop large multi-screen *single* video experiences on custom hardware. Dataton Watchout costs approximately forty thousand dollars per installation, which makes an inexpensive alternative appealling from a creative standpoint. ScreenPerfect permits people to use existing hardware to synch multiple videos to one set of controls. This is also distinct from ChromeCast, which allows people to wirelessly pair a television with a touchscreen for control and consumption of the touchscreen at a larger size.

Neither of these software packages provide any kind of support for a branching video experience natively, nor do they provide the ability to use existing hardware with same. ScreenPerfect provides this ability, because it has evolved out of the independent games community, rather than from the perspective of people who primarily consume television as a media habit. It is predicated on a comprehension of gaming and interaction that includes the ability to direct one's narrative, where the appeal of media is the appeal of *engaging* with media, rather than simply absorbing what an author—director has to say.

2.3 Theory

2.3.1 Helene Cixous and the *Écriture Féminine*

Cixous' *Laugh of the Medusa* predates the computer age, but perfectly and predictably describes the trouble with programming - which is a form of writing - within *Laugh of the Medusa*:

> And why don't you write? Write! Writing is for you, you are for you; your body is yours, take it. I know why you haven't written. (And why I didn't write before the age of twenty-seven.) Because writing is at once too high, too great for you, it's reserved for the great-that is, for "great men"; and it's "silly." Besides, you've written a little, but in secret. And it wasn't good, because it was in secret, and because you punished yourself for writing, because you didn't go all the way; or because you wrote, irresistibly, as when we would masturbate in secret, not to go further, but to attenuate the tension a bit, just enough to take the edge off. And then as soon as we come, we go and make ourselves feel guilty-so as to be forgiven; or to forget, to bury it until the next time. (Cixous, 1976, p.876-877)

"Write, let no one hold you back, let nothing stop you: not man; not the imbecilic capitalist machinery, in which publishing houses are the crafty, obsequious relayers of imperatives handed down by an economy that works against us and off our backs; and not yourself." (Cixous, 1976)

In this passage, Cixous chides her readers for not giving themselves the permission to write freely, or to be creative. French, Cixous worked with Lacanian theory, with many disciplines related to sex, which can be distilled to reproduction if one chooses. I do not so choose: Cixous wrote just as the pill was becoming available. Sex suddenly freed of the commitment of children by the first cyborgs, creativity - the act of creation - can now mean so many different things.

The guilt remains, though. Creative practice is difficult, and every new creative practice - programming, video art, game design - must go through the same flailing critique of its status as art, or as real at all, as the last new thing. The critique then works to isolate new creative workers, making them unsure as to whether what they produce can be considered work at all.

"Today, a software program or platform, once written and deployed, relegates its user to simple read/write tasks, with little use for changing the structure of the platform, and no ability or rights to do so." (Pepi, 2013)

"Simultaneously, the coordination of immensely esoteric skillsets are required to design and implement such platforms, consolidating power and capital with a small class of systems builders who may manifest their control in virtually any industry." (Pepi, 2013)

"Servers, broadband, hardware... the infrastructure of the digital economy is still closely guarded and accumulated by a shrinking roster of private interests." (Pepi, 2013)

2.3.2 History of Women in Technology

Ada Lovelace, daughter of Lord Byron, was the first programmer. She was excellent at rules, and beat herself up for it routinely Plant, 1997. The ability to put rules in order, to work backwards and forwards from a desired result all along the path of the machines, is a characteristic much sought in both programmers and game designers. Both roles are responsible for rule systems that will dictate a predictable result. Despite a historical involvement, women have been recently and quite comprehensively written out of technological roles.

The reasons for the write-out are clear. Partly, it is patriarchy. In a straighforward way, ladies may not possess uncomplicated positions of economic advantage within a patriarchy, and it is against the interests of the system to permit a polyphony of input at the rules-setting level. Computers have quickly become a good job with a good chance to better one's life. It is presently popular to assert that in the future, there will be two types of lives:

> "... those who tell computers what to do, and those who're told by computers what to do." - Marc Andreesen, Andreesen Horrowitz.

This seems broadly true, but something about the sentiment rings solutionist. Perhaps it is just that I do not personally like to think of a world where technology, and not humanism, drives society forward. I believe this is a swing state, and I believe it should be set aside, where possible. Anyone can learn to code. Learning to express oneself clearly in a creative medium is something harder.

screenPerfect is a tiny, didactic piece of software that only permits works within a specific framework. Like a gesture drawing, what is drawn is absolutely not implied, only the form. With that being true, it is astonishing how many works from the game jam ended up addressing questions of embodiment and stress situated within the body.

2.3.3 Lev Manovich, Alexander Galloway, and Software Takes Command

Alongside feminist written history, this thesis falls into the frameworks described by Lev Manovich in his 2013 book Software Takes Command. This book emphasises what Manovich sees as a gap in the academy's examination of *media* as the central component of art and literature, and seeks instead to directly address questions of how *software* can be itself analysed as possessing a direct impact on the systems of production with which it interacts.

I believe Manovich is overly aggressive in discounting the value and input of actual producers - I do not agree that individual forms of media are dead any more than I believe that print is dead - but I do think that his writing is directly related to my central research questions as to what impact a simple software tool might have on artistic prouction.

3

Software Design, Industry Engagement, and Hardware Design

3.1 Software Design

3.1.1 Interfaces, engines, and interactions

A software interface is the part of the software that a person interacts with directly **interactiondestext** where a software engine is the part of the code that detects and defines what a computer can do with that interaction. The engine that I coded responds to interactions sourced from users. Interface interaction is what appears to to define the majority of user experience, but the response of the engine underlying that interface is just as important. A key part of the design of screenPerfect is that it was laid out to handle things like auto-saving invisibly, so that a user's work would not be lost.

The interface of the software is just as important as the engine, however, because a poorly designed interface will confuse a user, thereby rendering the experience of using the engine figuratively opaque. screenPerfect's roots are as a software engine, which takes user interaction and then does things with it. The user interacts with the interface, which speaks to the engine, which then returns values to whichever interface the user has selected.

FIGURE 3.1: screenPerfect software communication model

3.1.2 Initial screenPerfect Engine—Interface Layout

In the case of screenPerfect, the interface is laid out in three parts. The first part is the setup screen, which is where game designers load their media (both videos and static

files) and lay out the links between those files. This is the essence of a game made in screenPerfect: which choice will a player make to navigate the system as designed by the artist?

The further screens are the client and control screens. screenPerfect supports up to ten client screens and ten control screens, although the interface only exposes a polyphony of client windows, while restricting artists to a single control set for simplicity's sake.

FIGURE 3.2: screenPerfect initial screen layout. Client, Control, Setup.

3.1.3 screenPerfect layout: NoJam

As the chief author of the screenPerfect software, it is very easy for me to understand where video files should go, and I come to understand how the software works implicitly. This is a common problem in software authorship, much as text requires editing and paintings require critique. I brought the software to Bento Box for refinement for reasons already stated: they needed a tool to express use cases for Daimio, and I needed help designing a UX that was actually useful to users.

The final layout of screenPerfect is much cleaner than the one with which I had been working. Rather than hidden tabs, everything is laid out clearly, and allows users to see where their files are going. It is open-sourced, and available on GitHub for evolution by advanced users - the DMG has already forked a version, iV (Faber, 2013), for further development.

What follows are screencaptures of the pre-fork game jam variant of screenPerfect.

FIGURE 3.3: screenPerfect NoJam screen layout. Client, Control, Setup.

3.2 Design Research: What Goes Into Starting A Software Project

3.2.1 Artist Collaboration

Beginning development from a first position within the arts is unusual, even for an Agile workflow, but in the case of this software, it has been wholly driven by artistic collaboration. I firmly believe that simple tools to relieve the friction points of the artistic process will lead to better art which is more widely available. This is to say, although the developer is themselves a creative who will decide *how* to solve problems,

the problems to solve may be better handled by an outside party. This is down to an issue of demand.

In order for software to exist, and to be seen to exist, it requires an interaction. Unlike a hammer, which takes up space on a shelf, software is essentially a text document until it is used. As Gallowaysays, a game is defined by action 2006. Galloway, 2006

3.3 Development Methods

3.3.1 Agile Development with an Artist Partner

The Agile methodology is based on a manifesto, as is so much else of this work. Agile is a response to previous software design practices, called "Waterfall," where software frameworks are laid out and heavily documented in advance of production. Waterfall methods are popular in major software companies, which rely on extensive documentation to communicate between business units. They emphasize planning over software production or delivery deadlines.

Agile, described in 2001 by a group of software developers, reflects a less top-down approach to the software development practice. Rather than pre-planning every element of software, screenPerfect was designed by discussion with key stakeholders as to how it should come together, and what the final product of the development process should be. This is a hacker-oriented means of development, reflecting Plant's statement that reverse engineering - "starting at the end, and then engaging in a process that simultaneously dismantles the route back to the start" is how hackers work Plant, 1997. Agile, released four years after Plant's assertion, engages the process of iterative development via reverse engineering in a more formal sense. Agile specifically emphasizes **individuals and interactions** over processes and tools, **working software** over comprehensive documentation, **customer collaboration** over contract negotiation, and **responding to change** over following a plan.

In the case of my development process, this worked as follows: The initial project was laid out by Hannah Epstein, who described how the game processes for psXXYborg should work using a series of YouTube videos linked through their annotation technique. In development conversations, it became clear that Youtube, in addition to having many distracting advertisements, also did not have a great way to make annotation invisible, and was very slow to load. This is a problem with reliance on external networks: they cannot be as fast as locally served files. Hannah specifically emphasized speed, smooth

loading, and using video based in static rather than streaming or live files. These needed to be served within a closed environment to an attentive audience.

From that point, I began to research languages that emphasized speed over structure. Scripting languages, without strong classes or inheritance, are especially good at this type of development work. I reached out to other developers and asked how they would solve this problem, and they came back to me with a variety of answers - some used PHP, some used Python, all of them relied on JS for their front end. In researching different ways to solve the basic problem - passing a variable back and forth through wireless technology to select two on-screen videos at almost the same time - I discovered the Node framework. I explained how this would work - minimum installation time and expense, reliance on a straightforward machine installation on-site - and got approval from my partners.

At that point, I went home and wrote a server using Express.JS, socket.io, and node to write an application intended for the Safari browser, in line with my partner's habit of exporting video in the restricted H.264 format. Due to conflicts relating to codec patenting, one of the many secret things that underly the "free" internet, Safari supports H.264 where Chrome does not. Chrome supports Webm via the V8 engine, the same engine that supports the Node framework. Webm is also a more compact video format, which results in smaller file sizes and lower bandwidth costs, which eventually affects both load time and playback lag on client machines.

Having worked extensively to serve H.264 video and discovered that loading many videos in H.264 will quickly overload the Node-native server, I reencoded the video works into the Webm format and converted my development process to pursue the Chrome browser. This also made psXXYborg available on mobile browsers on the Android platform, which is advantageous, as Android is much more readily available to low or no-budget projects than Apple devices, even old ones.

Throughout this process, I would meet with Hannah, who would provide me with updated feature requests. This is normal for the agile process, which places an emphasis on completing software delivery by deadlines, rather than on documentation and a firm attachment to original plans.

In order to keep track of all of the changes to the software, I used the GitHub source control system. GitHub allows users to store and update their code base while keeping track of changes in what are called "commits." Other GitHub users can then "fork" or copy a specific version of the existing codebase, and from there make their own changes. A "fork" is considered a new project. A "branch" is a "fork" within a project that contains changes not yet passed to the "master" branch. My GitHub commit log for

the screenPerfect fork of psXXYborg can be seen in Appendix C, which documents the changes made over months - bug fixes, design shifts, changes in the layout of the program, and new ideas for the setup and control files, including the late inclusion of how branching narrative would actually work without a database.

Overall, Agile worked for this process by allowing me to respond to user requests for code changes and information rather than forcing me to work to a standard pre-set from above. A Waterfall process would have discouraged me from even attempting to take on the work. By working in small steps back from a pre-set destination with total freedom as to how the code actually came together, Agile allowed me to demonstrate different working parts of the software as they came together. The documentation for the project is tied into the code commits, and inseparable from the actual written code within its archive.

3.3.2 GitHub and Open Source Software

The development of screenPerfect is dependent on a variety of external technologies. Although relatives and derivatives of Google's V8 system are foremost among these, there is also a dependence on the licencing and mindset of the open-source movement, and the GitHub software repository system.

Open source software is not the same as free software, as provided by structures such as the GNU General Public Licence . Open source is the peer reviewing system of software. What open source means is that even if a given piece of software compiles to a single program which can then be distributed for use on the desktop - as screenPerfect does not - the code that goes into the executable file is freely available on the internet, to be changed, supported, and developed by the population of software workers who exist in the broader world. These developers may work on closed or open source projects in their usual working time. They may be very skilled or quite new to development work. What matters is that the software's code is then shared publicly, where it can be reviewed and compiled and extended and changed by anyone at all.

The intent of open source is that anyone may learn from such freely-shared information, and anyone may contribute to the collective knowledge base. In this, GitHub is not unlike JSTOR. The stark difference is that GitHub is costly to the user who is sharing information, rather than to the user who is seeking information. A public GitHub account costs nothing, a private one with a limit on projects is less than ten dollars per month, and any public projects can be found online by anyone. This is useful for reference, as one can quickly find the solution to many coding problems for the cost of looking them up, and an internet connection on which to do so.

There are some obvious problems with open source. One of the clearest is that with all that intellectual property out there for free, it is a challenge to make any money on an open project. The other is that there is no way to guarantee quality: one takes what one can get, although it is assumed that contributions to projects are made in good faith, and major project contributions are checked by trusted individuals before they are published. For example, the Mozilla project relies on contributors whose code is applied to the codebase after approval by certified reviewers bbondy@mozilla.com, 2013, URL.

3.3.3 Licencing

One of the ways these problems are dealt with is through licencing. The Creative Commons at creativecommons.org expresses their mission as follows: "Creative Commons develops, supports, and stewards legal and technical infrastructure that maximizes digital creativity, sharing, and innovation." It is therefore an appropriate open standard licence for *creative practice*. A preferred licence for software development is the MIT Licence, which is closer to the Gnu Public Licence, but does not preclude making money from one's open source work.

The iV engine, a fork of screenPerfect, was resea

3.3.4 Science Fiction Inputs

My own idea for how this project would work is taken from Cory Doctorow's *Pirate Cinema*, which features a scene wherein characters climb trees, and using pico projectors already built into their phones, assemble a movie theatre from nothing more than sheets and ropes in the trees. I felt this sort of mesh-networked sharing is much more likely than a continued reliance on the surveilled internet for sharing copyrighted and copyrighted-material derived works. Since I could not find a system that would permit this type of sharing on the internet, I felt that this project would provide a good chance to build one.

3.4 Industry Engagement

3.4.1 Game Jams, A Design Method

A game jam is a variant on the hackathon, which is a type of prolonged effort at taking an idea from concept to finished product in a limited period of time. They are related to

design charettes or *parallel prototyping* (Martin, 2012), a method whereby participants rapidly prototype a design idea over a short, intense period of time. A jam - or hackathon - gives registered participants a common area and space to set up their own supplies, and a theme. The group members come to the event with an idea and possibly some resources - video files, sound capability and so on - and use the jam time to assemble a game.

Generally, a game jam will produce a panoply of small game ideas with fleshed mechanics but simple art and sound design in order to demonstrate a possible path forward for a device or piece of software, which will then be polished at a later date, and presented to the indie community either online or at a social event. Sometimes these works will then go on to be finished commercial products, or intended for further consumption at major conferences such as Indiecade or GDC. These conferences ideally further the careers of the developers by providing access to funding bodies: publishing houses, or in Ontario, the Ontario Media Development Corporation.

Game jams can be time consuming to prepare, as they involve a great deal of communication on the part of the show-runners. In order to run a jam, one must open the application period well enough in advance to ensure a large available population of skilled users who are likely to be interested in producing content with the available tools, or interested in exploring new tools on offer. Typically, jam members have a theme suggested - "Mother May I" or "Snacktember" being a few run in 2013 by the Dames Making Games - and then participants bring their own preferred technology to knock out a fast prototype over a weekend.

3.4.2 Dames Making Games and Game Jams

> Dames Making Games (or DMG Toronto) is a non-profit community organization based in Toronto dedicated to supporting dames interested in making, playing, and changing games. In short, we want to build an **inclusive** and **engaged** local community of game-makers. Our community isn't women only, but it is women-driven. *from the DMG.to website, accessed November 27, 2013*

The Dames Making Games are a society within Toronto that work to promote women in video games. They are funded in part by **FiG** and in part by member donations. I am an original director and advising director with the organization, which has given me ready access to a test audience for my ideas with regards to development tools. The Dames Making Games use the game jam method to introduce women and allies to simple game development tools. This provides a straightforward introduction to

concepts of computer logic and programming problems for some people, to video game art development for others, and video game sound production for still others. Some develop system mechanics, some design whole levels or game narratives.

The point of the DMG is to promote access to this field to people other than the 18-to-35 year old males who form the primary demographic for the video game industry, in the hopes that a diverse population of game makers will produce a diverse population of games.

The Dames Making Games are interested in screenPerfect as it provides an underlying template for a game-making system that might be easier for newcomers to use than the typically available game engines. The other two members of the Board of Directors of DMG are Cecily Carver and Jennie Faber, who, in exchange for development work as members of Bento Box, have since forked screenPerfect to become a more elaborate engine, called iV in honour of the idea of a Twine engine that created branched narratives from Vine videos. Carver, 2014, Klimas, 2009.

3.4.3 Bento Miso and Bento Box

Game jams require both space and people who are interested in working on games. A themed game jam, such as No Jam 2, which is designed to test specific software, requires a specific audience and some support to ensure that audience is interested in the work a researcher is doing. In order to access that space, I worked with the Bento Miso coworking facility here in Toronto, with OCADu's game::play lab and Emma Westecott, and with Bento Box, a development company that runs Bento Miso as a not for profit co-working facility.

Miso is a not-for-profit bricks and mortar site that serves as home for both Bento Box, a local development hub, and the Dames Making Games. It is also the hub of a great deal of Toronto's independent game development community. Miso/Bento Box offer professional support and development advice to game developers, and I felt there was a good match between their professional skillset and my research interests. The Dames Making Games regularly run a jam in November, and felt that screenPerfect - a new software designed to be accessible in a short time frame to people with extant skills - would be a good match for the audience associated with the organization.

Bento Box was also at the time seeking an engine that could display the capabilities of their private computer language, Daimio, which offers users the ability to reprogram work on the fly in the browser without being a trusted network source. Therefore, I

accepted their help and their offer of hosting the jam in return for giving them permission to fork - copy, reproduce, and extend - my engine under their name.

Miso, and Bento Box, offered to help me with coding a more accessible front end to the screenPerfect engine in time for the jam, so that I could get feedback on the system mechanics rather than just the interface.

3.4.4 NoJam 2: Video Video

The DMG have a great deal of experience running jams, and therefore, I partnered with DMG/Miso to get access to a group of skilled animators, filmmakers, and gamemakers. By partnering with them, I gained ready access to their community population, and they gained access to my software. One of the most common difficulties with game jams is that the short timeframe can cause a lot of frustration to new non-programmers: they spend a lot of time wrestling with tools, rather than generating the content of their games. The DMG would like to make it more straightforward for their membership to generate games and interactive narratives in a short period of time.

No Jam is a two-week jam scheduled by the DMG in November. In order to prepare screenPerfect for the jam, I handed over the basic engine to Bento Box - the production arm of Miso - who cleaned the interface elements and released a web-based version of the software for users. This was a win for them, as they were able to refactor my local code base to take advantage of a new language they have produced, called Daimio. Daimio, being a dataflow language, is ideal for describing choice patterns as they relate to a database. ScreenPerfect is a good engine match for types of games that rely on interactive choices.

As a pair, Dann Toliver - architect of Daimio - and I worked together to clean up the javascript elements of screenPerfect for speaking to the Daimio dataflow language. The group then released a refactored version of the code in time for No Jam, so that our participants could get a clean version of the software to work with. This was challenging for me, as it involved a great deal of trust, and moved the software away from how I had initially envisioned the UI. In particular, we needed to scrap an early idea for a branched narrative "tree" display, which was not included, although it had been planned all along.

After we received No Jam applications, we went through to choose participants who seemed interested the theme and the software restrictions, sent out acceptances, ordered food, and generally set the dates. Applicants were provided diaries to record their working process over the course of the week. The first weekend of the jam consisted of workshops from a variety of specialists to provide direction in how to think about the

software and the jam process as research. My presentation is included in Appendix C, consisting of how to work with the screenPerfect software, how to think about multi-screen video, and how to think about technology as a form of creative practice which is both limiting and freeing.

The applicants were then sent home for a week to work on their video projects, and asked to document their ongoing process with one another on a private Google Group. Most participants ignored this request, which left us with relatively little promotional material.

On the actual weekend, we asked that participants arrive with the majority of their video content and design prepared. There were vastly uneven responses to this request, which strongly affected the ability of participants to produce a finished game by the end of the weekend. I interviewed each group early in the process, and then later polled them with informal questions regarding their experience with the software.

The group experience with the software proved interesting. Accomplished filmmakers had a better time with it, but the most suprising response was from young, self-identified gamemakers, who rather than exploring what was possible within the context of the software tools, decided instead to try to use them to reproduce existing game types, many of which were totally incompatible with the software's design. Of particular interest was the group who tried to reproduce a classic Japenese roleplaying game within the context of video: this did not work so well, and they continued to work at it even after it became apparent it was unlikely to go well. The game itself remains unfinished, but deserves mention as the most unique and possibly stubborn effort. Used to working with uncooperative tools, the participants seemed unsure how to cooperate with a tool clearly designed to a single end.

Despite this surprising result, No Jam was a success, with nine groups producing diverse works on ideas such as how to express a practice of mindfulness, how to work with pornography in a way that forces the viewer to interact with what's happening on screen, exploring systematic violence against women, exploring narratives of imprisonment, magic, and in one unique case, permitting a puppet to escape a toy box.

In setting up No Jam, we did present at least one workshop on the importance of the personal narrative in producing creative work, which may have influenced the results. Game jammers mostly described their interest in producing work that was finished, and one jammer explicitly stated that she was pleased to have had a finished work at the end of the jam, this being an uncommon result for her when she had to learn the usual round of new software each time.

No Jam resulted in at least five "finished" works, which have since been included in several exhibitions around the city, including the December and January Toronto Long Winter series.

3.5 Hardware Design

There were many hardware difficulties associated with early builds of screenPerfect as psXXYborg. The software is dependent on open wiFi and high bandwidth, a stable computing system, a variety of tablets and other devices, and none of these things are confirmed to work together. Idealistically, the software is open. In reality, it is incredibly difficult to build a new software system to work on broad platforms, and this ended up being an unrealistic goal.

After installing PornGame at the Art Gallery of Ontario for a Long Winter event, even the software developed by Miso failed repeatedly to load correctly on a variety of devices. This resulted in the need to reprogram the psXXYborg build of the software to work within a limited network, and from there, I began to research what it would take to adequately display these time-based works. The answer seemed clear: A limited hardware system with a consistent environment, similar to a video game console such as the Super Nintendo Entertainment System.

With that in mind, I acquired a Raspberry Pi, a new type of microcomputer designed to be used for learning and prototyping new systems. The chief problems on display by screenPerfect were connecting at all, running the application consistently in hot temperatures or unreliable environments, and getting the various videos to display adequately on mobile and non-mobile systems. In addition, the many installations of various games in public began to imply that people love interacting with things, but that computers are unreliable in a public environment.

I observed that people are willing to use their smartphones publicly, but mainly to access the external internet, or messaging services while they are in public. ScreenPerfect, which relies on the form factor of a mobile device as well as a powerful multipurpose comuter, did not seem to work so well in this context.

This led me to consider how people interact with the internet publicly, and to consider topics of privacy and public space, and how these problems have already been solved by galleries and coffee shops wishing to offer their clientele data services to promote engagement.

In public spaces, internet is supplied by wiFi, which comes through a specific type of router known as a "captive portal." A user will walk into a shop, attach to a network, and "sign" an agreement to make use of the wiFi within that space.

Normally, the wiFi will then give them access to the external internet - the internet as supplied by a major ISP. This is not necessarily what needs to be supplied, however. In the Subnod.es project, hosted by Eyebeam in NYC, users pair to a captive portal which is also a server, supplying access to an entirely private chat room, which is available only to users on the network supplied by the captive portal itself.

This seemed like an excellent answer to the question of how to supply screenPerfect applications so that users can pair to them in an intuitive way, with a minimal amount of hardware that is easily maintainable, and I therefore set about building a Raspberry Pi that would supply two things: a wiFi signal and a server that serves an instance of screenPerfect where users can experience at least one, but hopefully more, screenPerfect games.

In this, I hoped to address a few problems. The first is that downloading applications to a smartphone seems invasive, particularly if those applications are experimental or site-specific, as - post-psXXYborg - I think that screenPerfect games are when they are at their best. The next is that web applications are very much not user specific - they can be experienced anywhere while they are on the open web, even if their content is intended to be restricted to a specific type of installation, or requires it for best use.

A small, portable piece of resilient hardware (a Raspberry Pi stores its entire operating system on a single SD card) seemed like an excellent answer, which would provide an appliance-like container to serve this software, and also provide a solution to the problem of external bandwidth reliance. By serving the application locally, there is no reliance on an outside pipe. A copy of the game can be sold, customised, and stored in a collection, if such is desired, or installed in any kind of specific cabinet for later use.

I decided on the Pi specifically because it is a cheap, accessible, reproducible system with a broad community of access and support, with the ability to include hardware controls where necessary. This type of system could also be built out on any leftover PC using a build of the Debian linux system.

3.5.1 Optimizing the Raspberry Pi

4

Physical Deployment of Web Applications in Limited Local Space

One of the earliest problems to come up in developing the screenPerfect sofware has been compromises in how the software is served to users and to artists. The reliance on the internet is difficult, because the internet simply is not always available - it is an unreliable resource that has come to be taken for granted. This is a problem of privacy, partly - anyone can be watched when they're online - but there are also simple installation problems.

WiFi is taken for granted in most institutions, but the fact is that access to these radios must be regulated so that users can be traced, and so that undesirable resources are not accessed. This limitation means that, for security reasons, it is rare that people can share new devices across industrial networks.

There are various circumstances in which one may wish to deploy a web application that do not include areas where a broadband connection to the larger world may be assumed. These areas include many music and art festivals. Any festival directly descended from the Burning Man network, for example, is unlikely to have a reliable connection to the outside world, the point of the events being to form a Temporary Autonomous Zone **taz** to explore broad ideas of art and culture. These zones may nonetheless benefit from art built to take advantage of contemporary smartphones, or contemporary radio technologies, all of which can be operated without a centralized power grid.

There are other challenges to public deployment, such as leaving a valuable production environment out in public, or requiring a technician to look in on specialized equipment. Both of these restrict the venues permitted for public display of work. Also limiting are instances where work should be deployed near people consuming alcohol, which is notoriously bad for electronics.

This chapter addresses my central idea on how to repair the gap between excellent idea for web-based deployment, and the physical reality of gallery spaces with sharply limited resources available for persistent software deployment.

4.1 Problems and Complications in Display

My method for discovering these issues has been to attempt to display variants of screenPerfect in user testing over the course of a year. We have had exhibits in a van, a tent, and repeated installations at various public bar events. These have been dangerous to electronics. As screenPerfect was developed and extended, and especially after an exhibition at the AGO which resulted in bandwidth throttling, the work began to point in a clear direction: screenPerfect needed to be an arcade box, similar to those used for years in fighting games in bars.

FIGURE 4.1: psXXYborg van installation

FIGURE 4.2: Street Fighter 2 Cabinet

Some brainstorming and a very hot summer day resulted in the following scenarios that a reasonable arcade machine would need to address in order to present advanced work to the new, highly exclusive exhibit scenarios.

Here are those design constraints.

1. There is never any external data service available. In the case of environmental exhibits, or site-specific installation, this is due to location. In the case of formal galleries, it is because bandwidth is not free, or locally reliable. An institution may only have 802.1x limited networks, as in the case of OCADu, or they may simply not have the money to supply thousands of gigabytes of data over the course of an exhibit's lifetime. This must be addressed, as a reliance on the outside means that the art exhibit cannot be reliably collected or maintained.

2. The exhibit is assumed to be public to anyone, which means that expensive equipment may go missing, or be broken. Art exhibits are often displayed in areas with a large audience walking around, and electronics are both delicate and worth money. Electronics purpose-built for exhibition should be easier to install securely in a given context, and harder to sell for parts.

3. The environment is assumed to be meterologically hostile - hot or cold, wet or very dry, and to be hosting at least one party, such as an art opening, possibly

with music. Exhibits outdoors or in new venues, such as Burning Man, are in environments that are terrible for electronics. It is good to put some effort into considering what kinds of devices can survive a rock concert while still displaying the artist's application.

4. The exhibition is assumed to be supervised by people technically unqualified to maintain a full computer, such as art curators. Building computer is a specific trade, as is the curation of artwork. They do not necessarily overlap. A new device should be so easy to use, it runs on being plugged in, and can be trusted to start back up again without trouble when something happens to make it turn off. Any new artwork's support materials should reduce, not increase, the burden of displaying that work, or the work may not be collected or displayed simply due to the massive skills gap between technician and curator.

5. The emphasis of the work should be on the work's display, rather than on a laptop screen. Audiences can be safely assumed to have seen a computer, or a smartphone, perhaps for many hours in their workday. It is therefore preferable to increase the scarcity value and absorption in the work on display by removing the emphasis from a familiar object, such as a laptop, to one with more aura, which could be almost anything but a screen of the standard sort used for brain work in North America. This will improve the value of the work by granting it an air of the exotic, the specific to a single experience.

6. The collectors of the work are assumed to have extremely limited resources for ruggedized workstations, which are costly. Ruggedized workstations, such as would be appropriate for remote environmental display, are costly, because they are designed for *doing* work, rather than *displaying* work. Ruggedization as a design practice stems from both practical considerations, such as bad weather, and a design language derived from military applications. Displaying work using a ruggedized machine brings in both a question of financial access, and an implied language of force which may not be appropriate to the work on display. It is therefore best to separate the specific necessity of waterproofing, dustproofing, impact resistance and heat control from the ideas visible in the typical ruggedized case.

7. Any data carriage for external connection - wiFi - is assumed to be overloaded by default. Art exhibits generally have at least one well-attended party associated with their opening, at which the gallery connection to wiFi is very limited. WiFi is understood to be the "free" access to the internet. For the purposes of this work, it is simply a radio technology. Because the main radio will be overwhelmed by many connections to the outside world, it is best to set up your own network,

> or captive portal, to serve the information you need served locally, rather than relying on a common carrier for the inside and outside world.

These are all very real constraints that heavily impact the collectibility and display of new media art, most of which is simply difficult to get out the door. We use computers for work and play, but we still separate our lives into periods when we pursue one or the other, and we still have boundaries between our personal and public lives. To use the same machines to display art as we do to *build* the work is to reduce the work from something approachable to any other tab in a computer. New media works especially must be seen within their exhibition context to be understood.

Julia Kristeva addresses this in her essay "Powers of Horror," where she confronts the issue of liminality, and how it holds people's attention

4.2 Subnod.es and Public Private Space

This project has a precursor using similar technology built at Eyebeam in New York in 2013. Subnod.es uses a captive portal similar to my own, based on the inexpensive Raspberry Pi framework, to display a chat client to only the local environment. The differences are substantial, although mainly located within the code. Subnod.es relies on an external DNS being made available via the actual subnod.es software, and depends on a different collection of software to serve the portal proper. It is also built such that those library dependencies are inseparable from the main project **subnodes**

The chief concern of subnod.es I have not yet mentioned: subnod.es was built as a response to concerns about communications privacy in North America under the NSA. Specifically, the author is concerned that people behave differently when they are watched, a subset of the concerns generally associated with panoptica and totalitarianism. While I have not specifically structured Captive Art Portal to address these concerns, it *has* been built to be largely private. It serves an application to a limited selection of a public space.

The assumption of the Captive Portal Art Machine is that galleries have limited resources, but that people who go to art galleries almost certainly have access to a smart phone, which is a form of private space. Smart phones are people's own homes, and are built to assume that they will stay with their owners at all times. This means that to install an *app* is to ask a lot of a viewer: specifically, it is to ask someone to bring an application into their private space without getting to sample it first. To contrast, serving that same application on the *broad* internet is to entirely delimit the context the

art may be experienced within, which reduces its scarcity value to almost nothing while simultaneously removing the curator's ability to set the context of an exhibition experience. This means that it's unlikely an artist can be compensated in any conventional sense, despite their large audience, and also means that the curation of the exhibit is no different than the "curation" found on Tumblr. Neither of these are fantastic outcomes.

A better outcome is to make a limited public space available in a private context, and this is what we are doing when we ask that people open their phones and look at a website. The Internet is, famously, the new public space. By presenting a web application using public technology within the exclusive context of the gallery - or desert, or forest - we take control again over how our art is presented, and from there, how it can be consumed. A gallery or exhibit space can be set up very specifically for the benefit of an audience in a way that the internet in general cannot be, and web technologies are uniform and affordable in the way that more custom projection design software is not.

This sense of limited private space is key to the code-switching that human communication relies on. We are not the same people in public as we are in private, and we are again different people when we are in different publics, work to the street to school to the gallery. Technology that sensitively addresses these different code contexts seems likely to benefit its authors and its users both.

4.3 Materials and Supplies

This entire build is predicated on a Raspberry Pi running a custom distribution of Debian linux.

Raspberry Pi
: The Raspberry Pi is a full linux computer the size of a large credit card. They run Debian linux off of a common SD card.

32Gb SD Card for Raspberry Pi
: This is where we place the operating system and software for the Pi.

USB wiFi dongle
: Edimax-based wiFi USB dongle, for serving wiFi hotspot on the Pi.

USB flash memory
: For transferring or storing complete programs authored on external systems.

Keyboard and Mouse
: For initial computer setup.

Ethernet Cable
Standard cat5 ethernet cable for programming remote.

HDMI TV and cable
Used as a monitor for the Raspberry Pi.

Micro USB and power supply
Power for the pi.

Mac or PC computer with USB ports, ethernet port, SD Card reader
Required for raspberry pi setup.

4.4 Background for Linux Commands

`sudo` is Linux for Do It Now.

`apt-get` is an inherited "package manager" from Debian linux. "Dependencies" are the software your software requires to run, Debian uses `apt-get` to manage them.

Things that follow `sudo` are commands.

4.5 Setting Up Your Raspberry Pi

4.5.1 Windows 7 SD Card setup and first boot

This section is written for a Windows 7 environment, and is based on the common tutorial at `http://learn.adafruit.com/adafruit-raspberry-pi-lesson-1-preparing-and-sd-card`

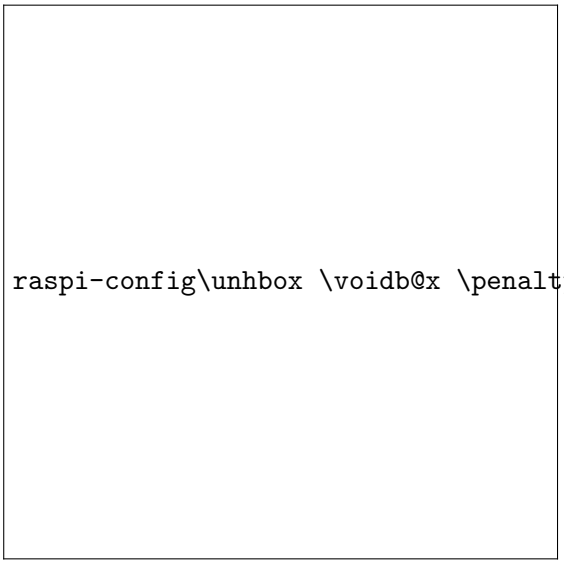
1. Connect your main computer to the internet.
2. Download the most recent Raspbian distribution image from `http://www.raspberrypi.org/down`
3. Download Win32DiskImager from the greater internet. This is preferable because it allows you to write image backups to your harddrive.
4. Using Win32DiskImager, write your Raspbian distro to your SD card on your main computer.
5. Eject the microSD card and stick it into your RasPi.
6. Plug in your keyboard, and plug a mouse into your keyboard.

7. Plug in your HDMI cable and monitor. Turn them on.
8. Plug in the MicroUSB cable for power to your RasPi.

4.5.2 Configuring Raspbian

Once the RasPi is turning on, it needs to be set up to include all of its software. Turn the Pi on, and wait until the blue configuration screen comes up.

FIGURE 4.3: Early RasPi Configuration Screen



Ω_expandΩ_1.png Ω_expandΩ_1.png

1. *expandrootfs* Expand the boot system so that you will not run out of onboard memory for software.
2. *memorysplit* Reduce the GPU to minimum, because we will be using the raspi as a headless server from the command line.
3. *changepass* Change the password so that your raspi will be less easy to hack.
4. *ssh* Enable SSH so that the pi will be accessible from an external computer.

When done, select `finish` to exit.

Type `sudo reboot` to restart the raspi.

4.6 Software Setup for External WiFi Access

A wiFi antennae can be used for one purpose at a time: it can either be used to access the external internet, for acquiring software to install into the raspi, or it can be used

for serving a hotspot. It cannot do both at the same time. To load the pi up requires external access, so we will be loading that first. You must configure your wiFi before plugging in your wiFi antenna.

In Linux, nothing will warn you if you mistype a folder name, say, adding an "s" to "network" to make it "networks." If you would like to confirm your folder name is correct, try typing "ls /etc/" to list the contents of that directory. Network is a default folder, and Interfaces is already present at first boot, so you can make sure your things are all there before you really get started.

The way to tell you have done something wrong is if you type the below command and an empty new file opens. You are editing a file here, not creating one.

At your console prompt, type the following:

```
sudo nano /etc/network/interfaces
```

This opens a text editor. Enter the following into it.

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0
auto wlan0

iface wlan0 inet dhcp
wpa-ssid "network name, commonly called an ssid, goes here"
wpa-psk "password"
```

Then type CTRL-X and Y to save your file.

```
sudo halt
```

Plug in your wifi antennae, pull your raspi's power cable, and plug it back in. This should make the raspi's antennae turn blue as it turns on. This little blue LED will frequently be the only way to tell something is going correctly or incorrectly, so it is an excellent tell that your machine is running.

FIGURE 4.4: Raspberry Pi with functioning wiFi antenna

withwiFion.png withwiFion.png withwiFion.pdf withwiFion.pdf withwiFion.jpg withwiFion.jpg withwiFion.mps withwiFion.mps withwiFion.jpeg withwiFion.jpeg withwiFion.jbig2 withwiFion.jbig2 withwiFion.jb2 withwiFion.jb2 withwiFion.PNG withwiFion.PNG withwiFion.PDF withwiFion.PDF withwiFion.JPG withwiFion.JPG withwiFion.JPEG withwiFion.JPEG withwiFion.JBIG2 withwiFion.JBIG2 withwiFion.JB2 withwiFion.JB2 withwiFion.eps withwiFion.eps

If all went well, you've now connected to your own supply of wireless internet. This will not work if you are using an 802.1x network, such as those within OCADu. On your own home network, however, type:

```
sudo apt-get upgrade; sudo apt-get update
```

This will upgrade your rasppi to whatever the latest agreed-upon package lists are, then update those packages to their most recent approved version.

4.7 Installing Node.JS

4.7.1 Why Node?

I've chosen to install Node because it is the software framework I selected to run the new game engine built in Part 1 of this thesis. Node is a new framework designed to get Javascript running on a server. There are advantages and disadvantages to this approach. The advantages are that JavaScript is a beautiful, minimal language that is relatively easy to learn. The disadvantages are that there is a heavy public bias against JS due to its years as a client-only language designed to manipulate what are known as Document Object Model (DOM) elements in-browser.

The brilliance of Node is that it replaces the need for a specific input-output window, replacing that definition requirement with any internet browser. Node, backed by Google's V8 engine, currently works best on Chrome, but it can interact with any browser.

Node is therefore easy to use, and easy to program for from the perspective of a mainly web based development chain.

4.7.2 Installation Instructions for Node.JS

Create a directory for Node to live in by typing the following at prompt.

```
sudo mkdir /opt/node
```

Acquire the node "tarball" - compressed framework files - via the internet.

```
wget http://nodejs.org/dist/v0.10.2/node-v0.10.2-linux-arm-pi.tar.gz
```

Unzip (desticky from tarball) it:

```
tar xvzf node-v0.10.2-linux-arm-pi.tar.gz
```

Copy the contents of the newly unzipped folder and paste them to your new directory. This leaves a copy of the tar and a copy of the unzipped tar at their original locations. You can probably remove them using sudo rm when you're sure everything is where it should be.

```
sudo cp -r node-v0.10.2-linux-arm-pi/* /opt/node
```

Edit - or create - a .bashprofile file, which is a type of script that runs when you turn on the pi. In this case, it runs and tells Node that it exists on your computer, so that typing node runthisprogram will do something. What is a .bashprofile?

From your root directory, to open a new nano text file:

```
sudo nano .bash_profile
```

Then add the following and save it to your new .bashprofile file...

```
PATH=$PATH:/opt/node/bin
export PATH
```

Control-X, Y to save it.

Node lives in the `/opt/node` directory you created above. This adds the commands "node" and "npm" to what are called "environment variables." If you are curious, and god knows you must be to play with a raspi, you can type `ls /opt/node/bin` and see the little programs sitting there in their bin.

4.8 Testing Node

Node will need to be able to fetch its own packages separately from the raspi from the internet in order to run some of the monitoring software I've chosen to use. Particularly, you will need the `forever` package.

4.8.1 Selecting Monitoring Software

`forever` has ultimately been the software I've decided on to monitor and run screen-Perfect, because it is a node-native package that keeps things running even when they crash. There are other software packages used for broader deployment, such as Monit, which installs to your Debian parcel rather than to Node. Monit typically runs with what is called an HTTP Proxy, which can be written directly in Node or installed independently. In a full deployment build, Monit and HAProxy would be preferable to Node

alone, because this follows the best practice of separating out different programming elements from one another in production. Monit and HAProxy can also deploy applications above and beyond Node itself, which is preferable for things written in Python, for example.

For this example, though, `forever` works well. It provides monitoring to tell us what the application is doing, and automatically restarts node applications when they crash. Were I deploying this such that it could keep an eye on the internet, which I am not, I would also include `nodemon`, as is recommended by the Subnod.es project. `nodemon` monitors your development code and pushes changes from a central server to your deployment automatically.

That is outside the scope of this paper at present.

4.8.2 Installation of Node Modules

To install a node package - or "module" - you type

```
npm install PACKAGENAME
```

To install one globally, type

```
npm install PACKAGENAME -g
```

To absolutely force install:

```
sudo su
PATH=/opt/node/bin/:$PATH
npm install PACKAGENAME -g
exit
```

To install `forever` and `nodemon`

```
npm install forever -g
npm install nodemon -g
```

To run `forever` and `nodemon` together....

```
forever start /usr/local/bin/nodemon /path/to/YOURAPP.js
```

4.8.3 Troubleshooting NPM installations

When I tried to install `forever` the first five times, it timed out, gave me a 404 error repeatedly, and declared I had insufficient permissions to do a global install. This is

where computer science faith, confidence, and patience come in. When the install did not work for half an hour, I took a break, came back, and discovered that it installed the next day.

This process is heavily dependent on a massive network of computers and other people. In development, it is quite likely things beyond one's own control are going to go wrong. Going for a break will help you keep patient.

4.9 SSH via Direct Ethernet Connection and WiFi Internet Access

Eventually, you will need both of the powered USB slots on the raspi for a USB key and for your wiFi. In addition, the raspi doesn't have the power to drive a monitor and consistently serve wiFi out of its USB ports. To get around this, it is most convenient to be able to SSH in to your device. Although it appears to be best practice to use the wpasupplicant file to store how you wish your raspi to connect to the internet, I have had limited success with it, likely because I am not configuring a static IP for my raspi properly.

My `/etc/network/interfaces` file looks like this:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
address [MY MAIN TERMINAL'S ETHERNET IP PLUS ONE]

auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
     wpa-ssid "network name here"
     wpa-psk "dubiously secure password"
```

```
sudo nano /etc/default/ifplugd

### MANY TALK, HOW COMMENT, SUCH WARNING ###
INTERFACES="eth0"
HOTPLUG_INTERFACES="eth0"
ARGS="-q -f -u0 -d10 -w -I"

SUSPEND_ACTION="stop"
```

This is an edit of the existing bits, and I can't tell if it will break everything long-term. Here is what your startup script should read. This ensures that your wiFi antenna

turns on, which is likely not something it was doing when you plugged in your ethernet directly.

```
sudo nano /etc/rc.local
#!/bin/sh -e

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

# Disable the ifplugd eth0
sudo ifplugd eth0 --kill
sudo ifup wlan0

exit 0
```

CTRL-X and Y to save, then sudo reboot open a terminal on your main laptop. On your laptop, at the prompt, enter:

```
ssh pi@[the static ip address you entered under eth0 static above]
```

Your `pi@[static ip]` should appear in your terminal window, which means you can now talk to raspi. Per usual, to ensure your wifi is still working properly, try a `sudo apt-get update` or `ping google.com`, both should return you data.

4.10 Backing Up Your RasPi

Now that everything has been configured for the first steps, type `sudo halt`, and when your raspi turns off, remove the SD card from it. Place the SD card back in your main computer and reboot Win32DiskImager.

Create a new file folder somewhere within your Documents folder. I called mine Raspberry Pi Backups.

In the Write From section of the application, select your SD card, which is probably called boot. In the Write To section, select your new folder.

Write a copy of the kernel image from the boot card to the new backup directory. Then safely eject your SD Card and re-insert it in the RasPi. It is best practice to form these occasional backups as you proceed through set up. Many of these steps can cause your raspi distro to break badly. A backup will save a great deal of time when the inevitable happens.

4.11 Mount Your USB Flash Memory Stick To Your RasPi

4.11.1 Configuring Your Mount Drive

This bears some thinking about, because the `/media/` folder is for media, and you are instead choosing to run a program off of the drive. Subnod.es suggests making it your `www` drive, for world wide web. I picked `/mnt/`.

Find your USB memory by listing the the things plugged into dev:

```
sudo ls /dev/sd*
```

If you've been following along, yours is almost certainly named "/dev/sda1".

So make a directory for it to be addressed at:

```
sudo mkdir /mnt/USBSTICKNAME;
```

Then mount it to that directory

```
sudo mount -t vfat -o uid=pi,gid=pi /dev/sda1 /mnt/USBSTICKNAME/
sudo reboot
```

Rebooting will restart the raspi but also close your SSH session. Watch the lights on the raspi board until they're stable again, about two minutes, then:

```
ssh pi@[static ip]
```

Oh look. Your USB drive does not automatically mount at boot. Problem.

4.11.2 How to Boot Mount External Memory

Find out the actual name of your external memory card:

```
ls -l /dev/disk/by-uuid
```

Write down the UUID of your USB stick.

This is the most manual way to run this operation, and there is software that handles automatic drive mounting. It is called `usbmount` and was discarded during this process because it ended up being more convenient to rely on my Node application being loaded directly onto the SD card, rather than from boot.

```
sudo chmod 775 /mnt/USBSTICKNAME
sudo sp /etc/fstab /etc/fstab.bak
sudo nano /etc/fstab
```

Add the following to `/etct/fstab`

```
UUID=YOURUUID /mnt/USBSTICKNAME vfat rw,defaults 0 0
```

CTRL-X, Y to save, then

```
sudo reboot
ls /mnt/USBSTICKNAME
```

This command should display the contents of your USB key when you go looking for it.

At this point, I have taken a copy of my Node application and moved it to the SD card in a separate directory. Although I have optimistically tried to make this a headless - no keyboard or monitor - box, realistically, lots can go wrong with the SSHing process. You will probably eventually want a keyboard, and it is much easier to store your access point as a single image per card, much like any other video game.

To store your games locally, rather than in the USB stick:

```
sudo cp -r /mnt/USBSTICKNAME /home/pi/YOURDIRECTORYNAME
```

4.12 Set Up a wiFi Hotspot

To get started, you will need some more software.

```
sudo apt-get install hostapd dnsmasq
```

When everything is done installing, you will be converting your `/etc/network/interfaces` file to serve a hotspot, rather than connect to the internet.

Here is what my final `/etc/network/interfaces` file looks like:

```
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet static
     address 169.254.222.xx #xx is a stand-in for an actual address, not included
    .

allow hotplug wlan0

## wlan internet connect settings are commented out for easy swap.
#auto wlan0
#iface wlan0 inet dhcp
#     wpa-ssid "network name"
#     wpa-psk "network password"

```

```
iface wlan0 inet static
     address 192.168.42.1 #42 is a joke about Douglas Adams, in honour of my
    thesis advisor.
     netmask 255.255.255.0
```

4.13 Configuring HostAPD

hostapd is the software that provides the access point using your raspi. It can be tricky, and in order to make it work, it needs to be compiled for one's specific model of wiFi antennae. For the purposes of this paper, we are using an antenna sold and supported by Adafruit. The appropriate compile of the hostapd software is included in the supplementary files to this paper, but can also be found at http://www.adafruit.com/downloads/adafruit-hostapd.zip.

To install a valid copy of hostapd:

```
wget http://www.adafruit.com/downloads/adafruit_hostapd.zip
unzip adafruit_hostapd.zip
sudo mv /usr/sbin/hostapd /usr/sbin/hostapd.ORIG
sudo mv hostapd /usr/sbin
sudo chmod 755 /usr/sbin/hostapd
```

Now set up a daemon - a piece of automatic system software - to run the hostapd configuration file on boot.

```
sudo nano /etc/default/hostapd
```

Uncomment (remove the hash mark in front of) `DAEMON_CONF=""` and replace that line with `DAEMON_CONF="/etc/hostapd/hostapd.conf`. Then type CTRL-X and Y to save your file.

My hostapd file is listed below.

```
sudo nano /etc/hostapd/hostapd.conf

interface=wlan0
driver=rtl871xdrv
ssid=piebox
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=berrybox
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

4.14 Configuring DNS access via `dnsmasq`

Configuring dnsmasq is straightforward. The installation package comes with an extensive config file, which lives at `/etc/dnsmasq.conf`, and includes all of the options necessary to turn on a DNS routing service.

To configure your dnsmasq installation, enter `sudo nano /etc/dnsmasq.conf` and then add the following lines to the top of the configuration file. The configuration file contains all these values commented out already, and may be worth a separate read.

```
interface=wlan0
dhcp-range=192.168.42.2, 192.168.42.50,255.255.255.0,12h
address=/#/192.168.42.1 #redirect all DNS requests to 192.168.42.1
server=/screenperfect/192.168.42.1#3003
address=/apple.com/0.0.0.0
```

What the above does is tell the raspi to listen on the wlan0 interface, to the dhcp range between 192.168.42.2 and 42.50, for twelve hours per time a client connects to the wiFi point. In addition, the portal is supposed to redirect all DNS requests - things like "google.com" - to the Pi's main address, which is - as we can see in `/etc/network/interfaces` - 192.168.42.1, and from there to the port 3003, on which my particular Node application listens.

In addition, the portal serves a spoof address to apple.com, which helps us to pop up the appropriate page on the captive portal when it is turned on.

At this point, reboot the raspi by typing `sudo reboot` and everything should come up properly and turn on appropriately.

4.14.1 IP Addresses

The trickiest part of this particular operation is the various IP addresses. IP addresses are raw internet magic. They are how we phone the internet, and how we talk to things once all the radios are tuned in, and they are admirable. From Wikipedia....

4.15 Setting Up init.d to Run `forever` as a Boot Script

5 Conclusion

5.1 Conclusion

The work that has gone into the production and release of screenPerfect is not inconsiderable. As a creative project based in grounded theory and reflective practice, code is a tricky thing to pin down. It must be declarative, yet it reveals the internal architecture of the people who write it. To write code is to be a craftsperson, and to be a craftsperson is to reveal some of yourself with every line. Programming solo is as rewarding as any other solitary occupation, yet it leaves many loose ends. An excellent piece of software is likely to require input from a wide array of specialists in graphic design, interface development, and logic. There is an inevitability to induced flaws - bugs - that cause the program to fail. Once complete, it is likely that finished software will fall out of fashion: software is the unfinished symphony of the 21st century. Just as there is no way to call a piece of writing finished, because another word can always be added or cut loose, code is subject to scope creep. Code changes. It is not a silkscreen, once pulled and forever finished. It is not a painting, which, dried and delivered, is safe until the conservators come for it. Code lives, like writing, in context and within an ecosystem. Unlike writing, code answers to its context; without the machines to whom it speaks, it is without consequence. Within those machines, it may have a concrete effect on the world around it, and for that reason it continues to be valued; this is the craftsmanship that, unlike art, continues to make a living. Code cannot be set aside; although it will work as intended, it will break without permission. To code is to attempt to write a coherent world into being, to attempt to factor in all the diverse chances within. In Cixous' Laugh of the MedusaCixous, 1976, she expresses that to be considered real, women must write for themselves. In my thesis, I have extended this to the world of code: one must write after one's own interests, because to take only that work which is assigned is to fall behind. Writing a projection/presentation/game system has been

work designed to address the problem of what, exactly, a game is, or what is a valuable piece of work. screenPerfect is designed to evaporate, leaving only the experience of its content to represent itself. It does not care what your content is, or to whom you're serving it, or where. The emphasis is on allowing your audience to experience things as quickly and easily as possible. Works produced using screenPerfect can be displayed anywhere in the privileged world; anywhere a series of screens and a single server can be set up. This emphasis on experience moves the interaction sphere into the world. The display of video-art and collaborative gameplay made possible through screenPerfect can be anywhere at all, and indeed works best at night, outdoors, in temporary installations. These are the new/old/new exhibits, the unmissable one-time-only parties, the experience that happens in a hard to access place but leaves no marks for future visitors to interpret.

The reflective portion of this research has been to address the question of use and pragmatism, as well as what constitutes research within an artistic context. The answer is difficult to quantify; research is pursuing a read, book-learning, critical examination of a practice, as all art is practice. Art is fundamentally blue collar, as is coding; both are works of craftsmanship, both can be helped along by automation, but ultimately, their declaration within a finished state is dependent upon the person who has produced them. The blue-collar trade of art has been sold out by the academy, linked tightly to the idea that thinking about a thing is more valuable than working on a thing oneself. Coders are legendarily difficult to organize, resistant to unions or to the thought that they are themselves labourers.

The newest video games try to dream worlds past being human, and most fall far short. Rather than permitting a wide exploration of possibilities, many possibilities narrow to the point of a gun, to the same forearms for twenty years (twitter). At the heart of screenPerfect is the idea that we can pull away from artificial distance and have instead on-site participation, unique experiences that project real, contemporary art into real, contemporary spaces. We can have events anywhere, and these events can bring anyone together, with even terrible technology. Underlying the architecture of this code is the idea that all the different flavours of classism should be undone with the opportunity to see amazing things, no matter who you are. Space should belong to the people who occupy it most often, not only the people who pay for it at a distance.

Learning new programming languages is always a challenge, as is enacting a pragmatic device from the perspective of art theory. There is the concern that these works are not for anything, not for a job or an application or a visible piece of content, a series of products released to do something in the world. Many applications, games and devices are released into the world that simply consume resources for the sake of their own

consumption, simply to show that the people involved have the resources to spend. A new digital toy is frequently out of reach for the vast group of people who depend on their existing technology to work for as long as they can make it do so, and therefore, these tools are designed in the same way as the first university mainframes: they live somewhere else, and can be accessed by even the most unfortunate smartphones, five years out of date and slow. The idea is to permit this sort of advanced media to get to places that I do not expect it to turn up, in places that are specifically not shiny. The idea is to use existing technology to permit exploration: use what there is, and then make it greater, rather than interject an external device that will require people to spend resources to use.

Ultimately, this sort of software development is about permission. Permission for people to do what they like in the spaces they need to occupy and use, including the digital space. This is not about developing a single app or a platform that will be easily marketed, but is instead about focusing on the value of the exclusive experience. The Game Jam, which is a located, people-in-the-room bit of enthusiasm, and the value of a small community which helps one another to develop, is reflected in the variety of games produced for the system. This is about exploring the possibilities of a space designed to share an experience through a personal connection.

This part is about the different values permitted to different classes of entertainment. At a high level, art can afford to be alienating, and indeed is frequently valued more highly for its power of alienation than for any other thing. This is a distinction made especially true in video games, where the power of "fun" tends to be valued highly for its commercial properties. If a game is not "fun," all elements of its interactive powers of storytelling cease. This means that games which are not "fun" are widely called by other names - interactive new media art, for example.

screenPerfect is designed to permit the development of games that are not only not necessarily fun, but which may be narratively incomplete, or nonsensical, while still being absorbing. These are games that do not require reading, but permit themselves to be experienced as deeply as a given group of readers wish to experience them. The tool can be perverted: perverse use is built in, with video copyright being at such a premium. It can be used to host experiences in galleries or in warehouses, by people with minimal technical knowledge and little ability to mask their normal online activities. This is a device to let people make maximum use of the devices they already have, to host a dance party on a subway or a massive art tour through a gallery. The demonstration content may be upsetting, but the access permitted is broad. This is about sharing things, for the better.

6 Endnotes, References, Works Cited

6.1 Works Cited

Anna Anthropy, Rise of the Videogame Zinesters, http://www.auntiepixelante.com/games/
Merrit Kopas, http://mkopas.net/ Zoe Quinn, http://www.beesgo.biz/ Zach Blas, http://www.queertec
bombs/ Porpentine, http://aliendovecote.com/ Soha El-Sabaawi, http://www.el-sabaawi.com/

6.1.1 Software

A

Agile Manifesto

A.1 Agile Manifesto

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more. © 2001, agilemanifesto.com

A.1.1 What Is Agile?

* Put in what is pair programming as well.

Agile is an ideal based on a manifesto released at the turn of the century. It is not specific enough to serve as a development framework, but instead serves to structure a way of thinking about software development in collaboration between developers and their user population. The original Agile Manifesto was released in 2001, and has a vast number of signatories to date.

Agile is intended to address the gap between planned software, which is typically fixed at release, and reality, which is that software breaks routinely and needs regular maintenance and iterative upgrades. The manifesto suggests that developers should focus on results over process, software that works over software that's well-documented, collaboration over contract specifics, and responsiveness to change over following a specific plan.

Agile has proven useful for this project because its emphasis on responsiveness and deliverables has ensured that the working software of screenPerfect can move forward to permit more games to be made, rather than locking the software to a concern for itself as an object. This is a valuable way to move through a development process which spares us from needing to commit to a given method, and instead emphasizes results from practice.

This practice must be documented, which is where the code-commit process appears. Agile deprivileges documentation for itself, preferring to include the necessary as the code itself is logged. Therefore, I have chosen to use the Git toolset to retain my version control on both this document itself and the code developed to run the main game engine.

B

Annotated Literature Review

B.1 Literature Review

http://kotaku.com/the-weird-escapism-of-life-sims-730629952 Leigh Alexander on aspiring to pay a mortgage in real life, which is important because people won't be able to do that, a lot, in North America.

http://agilemanifesto.org/principles.html The Agile Manifesto, which backbones my actual software development style: working software is what counts. Important, as Agile is in direct defiance of corporate control structures.

Maly, T. (2013). We Have Always Coded. Medium.com. https://medium.com/weird-future/2acc5ba75929 This article investigates gender essentialism from the perspective of biological essentialist arguments frequently used to say women can't or shouldn't code because of various, entirely specious, evolutionary problems. This is an argument that happens on top of a perceived resource scarcity; the scarcity is in this case employment of women in technology, so opportunity.

Fashion article about hegemony of fashion writing and the death of exclusivity, with notes on shows being locked down to resist the blog invasion. http://thenewinquiry.com/essays/cool-fronthot-mess/[I need some work on how basic economics works with supply and demand in the absence of a good money supply.

bell hooks. (1992). Is Paris Burning?. Black looks: race and representation (pp. 145-156). Boston, MA: South End Press. bell hooks is always useful in concert with Derrida to explain that you can't actually know what anyone else is actually thinking; having sympathy for other people is not the same as understanding their direct experience. Useful because it underlies a lot of feminist practice. This is an article about exclusivity,

which is an issue of privilege, which is an issue of perceived resource scarcity, in this case access.

Bizzocchi, J. Tanenbaum, J. (2011) Well Read: Applying Close Reading Techniques to Gameplay Experiences. In Well-Played 3.0, Drew Davidson Eds., Etc press www.etc.cmu.edu—well-read-jim-bizzocchi-joshua-tanenbaum Something to explain game design processes in the technical section of the document.

Buxton, W. (2007) Sketching User Experiences: Getting the Design Right and the Right Design. Morgan Kaufmann Publishers. Something to explain user interface design practices in the technical section.

Chun, W. H. (2011). Invisibly Visible; Visibly Invisible and On Sourcery and Source Code. Programmed visions software and memory (pp. 1-54). Cambridge, Mass.: MIT Press. Still need to read this, but the title is pretty relevant to the central research question of my thesis, which is on the value of code and invisibility as a permission.

Cixous, H., Cohen, K., Cohen, P. (1976). The Laugh Of The Medusa. Signs: Journal of Women in Culture and Society, 1(4), 875. Woman must write her own desires into being in order to be seen. This is a paper that reinforces arguments about scarcity of perception, and issues of control, specifically of women and women's opinions.

Deleuze, G., Guattari, F. (1987). Introduction:Rhizome. A thousand plateaus: capitalism and schizophrenia (pp. 3-4). Minneapolis: University of Minnesota Press. Everyone else is doing it [including them]. May as well. I gather they're popular right now.

Deleuze, G. (1992). Postscript on the Societies of Control. October, Winter(59), 3-7. Surveillance culture is bad for people, yet inevitable as it becomes automated. This is an issue of control, which is subverted by taking possession of even a single means of production; to refuse to code is to be illiterate of the systems by which production is governed. To refuse to acknowledge the implicit control of a system is to lie to oneself; if other people have designed the system, the system operates[Alex Leitch, 2013-10-01 2:18 PM There are no complete systems, particularly in computers, because Godel's incompleteness theorem says so. This is a joke that is also true and I am not sure how to cite it, but it holds for most systems of even informal logic, which computers are composed of. This is a purely theoretical way to look at a computer system that is both true and not true; the incompleteness theorem concerns math systems, not people, specifically. You can't test people for their incompleteness. But people are still incomplete. If they weren't incomplete, they wouldn't have religion.] as it is intended. The way out of this is to read the system, then find the gap within it.

Dell, K. (1998). Contract with the skin: masochism, performance art, and the 1970's. Minneapolis:University of Minnesota Press. Pain or revulsion is another way to escape a system, which is to make it so dear - dear as in price - that it is difficult to reproduce the work because it costs too much. Abjection, or the ability to pay for something with revulsion, is one of the less efficient but more effective systems of resistance. The liminal space represented by a willingness to publicly maim oneself is reserved for those who do not fit well within a system that relies on completion: broken skin stands in for a refusal to submit to hierarchy. This collapses in various ways over time - it's unlikely that even facial tattoos will keep people out of the workplace for long - but still represents a real way that people refuse to be included in a more perfect/uniform work. This is an article on how masochism, the revealing of the inside, has a recent history in art and what role that sort of display performance contributes to feminism.

Doctorow, C. (2008). Little Brother. New York: Tom Doherty Associates. A detailed look at how surveillance culture breaks down social contracts, and a how-to guide on resistance via action disguised as a novel. Also has a variety of excellent, accurate examples of ways to disguise data so that it cannot be confirmed by a rogue authority. Connected to Deleuze on Societies of Control, and specifically addresses homeland security spy tactics.

Doctorow, C. (2012). Pirate Cinema. New York: Tom Doherty Associates, LLC. Contains a scene with multiple tiny projectors used to set up a cinema in a park from pockets, which is more or less what the software itself is supposed to do when it runs. The entire book is about resistance to copyright authority. Contains a quite didactic passage on how even hardware control chips do not actually control or prevent smart-enough people from using controlled software, and in doing so, presents a vision of the world where the most privileged are no longer privileged with money alone, but also with knowledge or access, which is a type of prestige - which is a type of magic trick. Concerningly libertarian.

Gleick, J. (2011). The information: a history, a theory, a flood. New York: Pantheon Books. A survey text of the history and development of data-centric information technology. Explains a little of the context for how tools like screenperfect can be expected, themselves, to proliferate to the point of uselessness. This is useful because it prevents needing to look up each paper about completeness theories and map-rename signal-to-noise mathematics independently. Signal-to-noise mathematics are important because they provide a way to think about how to privilege information in the learning process, or the internet search process; people passively look up how to find what they

need. Downside: The noise often contains trace characters that allow further exploration. Upside: there really isn't that much signal out there, no matter how much noise is happening.

Gram, S. (2013, March 1). Textual Relations: The Young-Girl and the Selfie. Textual Relations. Retrieved April 12, 2013, from http://text-relations.blogspot.ca/2013/03/the-young-girl-and-selfie.html Young women's bodies are not only super-powerful, but can be the stereotype vehicles for all of consumer culture. This is important because young women are disproportionately discouraged from tech culture, even as they control the scarcity that is approved sexual relations within North America, a scarcity that cannot be overcome with money alone - you can't buy love, they say. So this is valuable because it is an excellent analysis of how stereotypically correct bodies benefit from fitting into a system of action, which is related to code practice because only stereotypically correct bodies are encouraged to participate. Per masochism, however, there are no correct bodies, only correct images of bodies. Resist the system in a predictable direction, and you become a new format of marketed body, with a new predictability. This predictability can be coded, but the closer one gets to perfect, the harder it becomes to occupy the role while remaining human. This is a deeply misogynist text, but is a perfect text for examining the role of image on the internet, and things which are seen but hard to describe, as code is.

Grosz, E. A. (2008). Chaos, Cosmos, Territory, Architecture. Chaos, territory, art: Deleuze and the framing of the earth (pp. 15-28). New York: Columbia University Press. Technology as sexual performance and definition of space. Not terribly well-realized but more academic than other sources on the same subject. Useful because code is a creative process, and creative processes - per Wilde - are useless ... like peacock feathers, or any other sort of look-at-me performance. Even things which do things are useless.

Haraway, D. (1987). A Manifesto For Cyborgs: Science, Technology, And Socialist Feminism In The 1980s. Australian Feminist Studies, 2(4), 1-42. Primary text on women restructuring their bodies, invisibly, to take over the world. See also Quinn Norton on IUDs (http://www.quinnnorton.com/said/?p=404) - this is useful because women can resist commodification, such as that described by TIQQUN, invisibly. Code is, in Agile practice, shifting from architecture to a sort of cooking; this library and that, all put together in a frame to pursue an idea, rather than to do a specific thing from the outset. This is a text about resisting control systems by allowing oneself to cooperate until there is a space to break free. Frequently, people don't even notice you have.

Haraway, D. (2009). The companion species manifesto: dogs, people and significant otherness. Chicago, Ill.: Prickly Paradigm Press. More Haraway. Now on cancer, not

sex. I need to read this but I don't think it will be too useful, except that it articulates that humans, with their tool-use, are not actually special; we are part of a system of mammals. This may be useful elsewhere.

Hunicke, R., LeBlanc, M., Zubek, R. (2004) MDA: A Formal Approach to Game Design and Game Research. sakai.rutgers.edu—hunicke2004.pdf More on game design techniques for the technical portions of the paper. Describes methods which will need to be addressed as part of the Agile/How Did I Make This Game methodology.

Kristeva, J. (1982). Powers of horror: an essay on abjection. New York: Columbia University Press. (http://www.csus.edu/indiv/o/obriene/art206/readings/kristevaHorrifying things have a power that is more potent than any non-horrifying things could hope to possess. This paper details why that is. It goes very nicely with Cixous and discussions of the IUD, because it is about what happens when barriers truly break down. I think Kristeva's horrors are basically the key to the entire news cycle and Grand Theft Auto to boot. This paper is the original on how revolting things are fascinating but resist being part of a system, unless they're cleaned away and perfect. See above comments on masochism paper; the awesome attraction of the awful.

Krug, Steve (2000) Don't Make Me Think: A Common Sense Approach to Web Usability. Riders Publishers. This is another technical paper for arguing that software design should be totally invisible. Useful because it ties together the systems of control argument - control is implicit, presented as undefeatable, a smooth surface - with the idea that things should be useable, so that people can find their own uses for the tool beyond what is initially intended by the author.

Schafer, T. (1998). Grim Fandango (1.0) [Video Game]. USA:LucasArts. Classic adventure game with minimal interface and a fixed runthrough. One of the last great adventure games. An excellent exercise in game design where the game itself is preset, but the ideas the game displays, including an interest in a subculture that is not much popularly examined (Mexico), and a good narrative. Evidence that narrative is important in gameplay, which is key to the development of screenperfect as a narrative branching tool. Also remarkably and incredibly broken on contemporary systems, as the initial code was rendered directly by processor speed, rather than at a stage or two removed; the game was broken by Moore's Law, which is good evidence for why tools need to be considered unto themselves. The new narratives are temporary. This is a narrative about the temporary; death, and the waiting period before leaving - while being wound up in a longstanding celebration.

Luvaas, B. (2006). Re-producing pop: The aesthetics of ambivalence in a contemporary dance music.International Journal of Cultural Studies, 9(6), 167-187. Retrieved April

10, 2013, from the Scholar's Portal database. An interesting look at what ethnographic research can be, and the speed of cultural shift and recycle since the rise of the internet. To be read in concert with various VICE mag articles about cocaine, new york. Used originally in article about Seapunk movement.

Moggridge, Bill (2006). Designing Interactions. MIT Press, Cambridge MA. More technical reading about how people interact with software, about how people can control interactions.

Moyer, J. (2012, September 14). Our Band Could Be Your Band: How the Brooklynization of culture killed regional music scenes - Washington City Paper. Washington City Paper - D.C. Arts, News, Food and Living. Retrieved April 22, 2013, from http://www.washingtoncitypaper.com/articles/43235/our-band-could-be-your-band-how-the-brooklynization-of/

Cultural uniformity because the internet makes things from different places seem the same, even though they're really not the same. Relates to the Young-Girl article about how if you are one perfect shape, that perfect shape will always sell at least a little, which obfuscates the truly beautiful and interestingly specific evolutions with things which have been data-optimized to be more popular. Popular isn't better, and neither is monoculture, but also no good is the sort of individuality that is itself a sort of monoculture.

Mulvey, L. (1975). Visual Pleasure and Narrative Cinema. Screen, 16(3), 6-18. On the male gaze, which is the central gaze in most videogames, particularly first-person shooters. This is important because the male gaze sets how most blockbuster video games are allowed to be perceived. Important because video games, like most software, are mainly compared to cinema, even though they have very little in common with cinema for elements beyond the technical. Core to arguments about how women are seen, which is essential to understand the TIQQUN readings in their slightly tongue-in-cheek misogyny.

One Laptop per Child. (n.d.). One Laptop per Child. Retrieved July 3, 2013, from http://one.laptop.org/ I was thinking about discussing how the OLPC project led to various other tech advances, including the rasPI - it made netbooks happen, then tablets happened. The OLPC was the project that said "wait, things don't need to be faster, they need to be better." Absolute disaster; in the countries it was intended for, it was already superceded by mobile phones. Classic example of condescending outsiders trying to Make A Difference rather than examining difference. Probably too broad a scope for this project.

Orlan: a hybrid body of artworks. (2010). London [u.a.: Routledge. Orlan led to Lady Gaga so directly that she has since sued her. Discussing the liminality and limits of flesh without Orlan's surgeries is a challenge; almost no other artist (burden? Shoot) has gone so far, but this distance is collapsed in film like Nip—tuck and the normalization of Hollywood surgery. Related to Kristeva and articles about the mortification of the flesh for the sake of appearances, which is what I am interested in with the arcade box. Although I want that to be subtly upsetting, not overtly upsetting.

Reines, A., TIQQUN. (2012). Preliminary materials for a theory of the young-girl. Los Angeles, CA: Semiotext(e) The new translation, which includes a feminist preface by Reines about the body of young women and how she almost was sick over the assertions of TIQQUN, which happened about the same time as everyone else was going bananas for Second Life, a game where you make an entirely new body that has since been abandoned by all but the most escapist. TIQQUN accurately observe that people are escaping into their own bodies, not those of the computer screen; the new presentation is that the brain and image on the internet reinforce the physical appearance through the phone, a piece of technology governed by the male gaze.

Stephenson, N. (1995). The diamond age, or, Young lady's illustrated primer. New York: Bantam Books. This is pretty well a perfect piece of fiction about cyborgs and universal education and China as an Oriental-escape paradise. Fun look at a post-scarcity economy that has simultaneously happened and can't happen. This is a book about an alternative resistance to the always-on personal presentation future, where books reflect their users. This is a fantasia, but an appealing one, with a lot to say about the subject of veterans, what abuse looks like from a perspective other than the dominant, and what recovery might look like. The main characters are all female, and all develop in different directions, including one who escapes by using the book to hack out a new life under direct supervision. Contains an unpleasant thesis about the value of personal matriarchal influence on future leadership.

Sternberg, M. (2012). They Bleed Pixels (1.0) [Video Game]. Toronto:SpookySquid Games. Excellent representation of a female lead game character in a genuinely challenging platformer. Useful because it exposes the programmer's preference for difficult-but-rewarding game mechanic loops, along with a conscious choice to show a young woman who has strong personal agency as a hero. A manifesto for better, simpler video games.

Swartz, A. (2013). Aaron Swartz's A programmable Web an unfinished work. San Rafael, Calif.: Morgan Claypool Publishers. The internet doesn't belong to us, but it could, and here are some technical guidelines to pursuing that as a worthy goal. This is the other way to approach technical development; something that should be extended rather than presented as complete in and of itself. Swartz rebels against societies of

control by describing systems to expose information at a basic level rather than obfuscate them. This eventually led to his death.

Team Little Angels (2009). Bayonetta (1.0) [Video Game]. Japan:Sega. What a hilariously sexist but also perfect meta-narrative of female power while subject to the male gaze. The rudest, most violent fun game released to ever feature a lady protected, literally, by her hair. A game with a strong female lead in the hilariously Kate Beaton "strong female characters" mold, which is problematic in its presentation even as it is simultaneously winking. Has unfortunately fixed gameplay goals, but allows players the reward of working through the game on a basic mechanic of style rather than skill alone. Fun!

Toom, A. (2012). Considering the Artistry and Epistemology of Tacit Knowledge and Knowing. Educational Theory, 62, 621-640. Retrieved April 12, 2013, from the Scholar's Portal database. More technical information on how to design interfaces so that people understand, passively, what they're supposed to do with it. This is about passive learning, which is how most people learn software: through exposure and experience with previous systems, we understand the language that the developers no longer expose even though help systems. The way of using the software becomes implicit.

Volition Inc. (2011). Saint's Row the Third (1.0) [Video Game]. USA:THQ. This and the followup, Saint's Row 4. Games that took the GTA pattern and subverted it to make a game that is cleverly and strongly and messily about playing video games and the fantasies of those games. Saint's Row is a sandbox video game about playing videogames and what a videogame means at its base. It allows people to play as whatever type of character they like, which exposes the fallacy that videogames are solidly about anything but mechanics; the art and design on top expose the code in their very mutability, but there are no ways to solve the game puzzles except violence. This is entertaining, because rather than being a game about traffic patterns and random mayhem specifically (GTA-V), it is a game about playing games, about false achievements and the ability to do anything at all as long as it's violent. Also notable because first lead female character is a hacker from the FBI. This is an important plot point. You can also play gay or with the robot AI you rescue ... but not the vice president, who's a dude. Central to my argument that software development is a second-stage creative practice because with no fixed skins, the game itself is much more exposed.

C

Game Jam Documentation

C.0.1 Questions To Ask Game Jammers

- What were you expecting when you came to the jam?
- What features did you immediately want in your software?
- How has your group process worked throughout the week?
- How is your group process going today?

C.0.2 Games List

- Porn Game by Maxwell Lander
- Grimoire by Katie Foster and Mikayla Carson
- Kill Fuck Marry by
- Mind Safe by Dann Toliver and Robby
- Glitch95 by Arielle, Rebecca and Bronwyn
- Omm by Brittany and Diana
- Cyborg Goddess by Cara and Kate McKnyte
- Empty Puppet by Danielle Hopkins and Dawn

C.0.3 Bug Discovery

- Room Zero must be the first room edited.
- Room Order cannot be altered in a meaningful way - ID is hidden from users

- WebM video is unplayable on Apple devices
- H.264 video is slow to unplayable on non-Apple devices

C.0.4 Features Requested by Game Jammers

- Sound effects on control input - requested by Arielle
- Timed hotspots which appear and disappear on specific video cues
- A game tracer that tracks which choices players make, and records their games
- Gesture controls - pinch, zoom, throw - on touchpoints.
- Tree View to visualize how a game is laid out
- Rooms cannot be deleted - delete and reorder rooms
- Games cannot be deleted - delete and reorder games
- Copy and paste room layouts so that one does not have to recreate grids - done.
- Hotspots that can move around the room.

C.0.5 Notes from committed jammers about screenPerfect

For Arielle, the most engaged of the jammers, the idea of turning any touch device into a custom console controller, with custom buttons, is engaging. The more traditional the game developer, the harder a time they had with the idea that they'd be showcasing content with the narrowest help from the new tool. The filmmakers were very impressed with the ability to not touch a darn thing and have considerable success.

Appendix D: Installing screenPerfect

D.1 screenPerfect Software Dependencies

- **Node.JS** - a software framework for writing javascript on the server
- **MongoDB** - a lightweight database framework utilizing JSON for file storage
- **Daimio** - a dataflow language that stores the game-states of screenPerfect
- **Git/GitHub** - version control software used to store and pull most recent versions of software
- **screenPerfect** - software for branching narratives in video and image

Detailed installation instructions for ScreenPerfect on PC

1. Install node.js http://nodejs.org/ 2. Install and run mondoDB following the instructions found below: http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/ 3. Either download and install Git/GitHub or download a copy of the SP working directory from pretentiousgit/screenperfect-dev, and install in local code directory. This should be somewhere that you are comfortable to find using the command line.

D.2 Running screenPerfect from a command line

1. From the command prompt, CD to the directory where you placed your copy of the screenPerfect files from web (for example, cd documents-dev). 2. Type node screenperfect at the prompt. The prompt should return:

```
screenPerfect is listening on port 8888
```

3. Open your browser and head to:

```
http://\textit{localhost}:8888/control will bring up a control interface
http://\textit{localhost}:8888/client will bring up a client screen
http://\textit{localhost}:8888/edit will bring up an editor, which will be only
    some use without a desktop.
```

D.3 Accessing screenPerfect from a remote client

D.3.1 On Mac

1. On your server machine, open a Terminal window, and enter 'ifconfig' at prompt. 2. ifconfig returns many results. You are looking for one that says 'inet.' The first ip address after inet is your localhost's dynamic address on your current wiFi network. 3. To find screenPerfect, enter this address - for example, (192.168.10.10) - then :8888. http://*192.168.10.10*:8888 should bring up the main screenPerfect game list.

```
http://\textit{192.168.10.10}:8888/control will bring up a control interface
http://\textit{192.168.10.10}:8888/client will bring up a client screen
http://\textit{192.168.10.10}:8888/edit will bring up an editor, which will be
    only some use without a desktop.
```

D.3.2 On Windows

1. Open a CMD window, and enter ipconfig at the prompt. 2. Take the number next to IPv4, for example, *192.168.10.10*. 3. Open your browser and enter http://*192.168.10.10*:8888/ to access the main game trees.

```
http://\textit{192.168.10.10}:8888/control will bring up a control interface
http://\textit{192.168.10.10}:8888/client will bring up a client screen
http://\textit{192.168.10.10}:8888/edit will bring up an editor, which will be
    only some use without a desktop.
```

E

Transcriptions of Public Talks

E.1 Transcript of PsXXYborg presentation, August 29, 2013

Rapture of the Nerds. I haven't had any internet at home for about a week. I figured that out today. This is what the van looked like, and we were all super proud of the van, which has gone to its final resting place.

So... all of mine is in text. I hope you like reading.... because what I do is text. I was not heavily involved in any of the theory parts of this project, or much of the artistic development parts of the process. I heard Feminist Art Game and Donna Haraway and I thought...MMM! Excellent! These are things I have studied! I have written a lot about these things! I can probably do this. And when I was invited to the project, we didn't really have a person to make the dual-screen part ... happen. And then we tried to hire a couple of them, and that did not go well. So the net result was that I would go over to Hannah's place, and I would hang out, and they would say "How are we going to DO this?" and I would say "aaaaaaauuhhhm, I've heard of this thing! Somebody told me about it last week, we'll use that."

This is not really an easy or sensible thing to do. Most people come to coding [a new project] already knowing a language, and they're comfortable with that language, and that language has conventions and assumptions, and ways it works. The language that psXXYborg is written in is Javascript, which until recently was not available for use on the server. It was only available for use in the browser. I'm sorry if none of this makes any sense to you. You can ask me later.

[Learning Javascript and writing psXXYborg in it] is exciting and very very new, and it was on the basis of a lot of money. Google has paid a lot of money for the technology that drives psXXYborg to exist, and it really wants us to use it. So I had heard of

Node, but I'd never really touched it, and I hadn't written more than about four lines of javascript in my life before we started this project. But I said "oh, we'll use Node and that will be fine." I think maybe Cecily and Jennie understand how much of a joke learning an entirely new computer language for a single project is.

psXXYborg is about new ideas, so it runs using new ideas. These are ideas about the direction of the web, and how to use it. Google has very specific ideas about how it would like the web used, and we used some of those ideas, which are about interactivity and the Chrome browser, to make this thing work. It works in ways that it shouldn't work. It works using a programming language that was not designed to build streaming web servers, and it uses the very newest ideas to do that, so a lot of psXXYborg was about research.

Coding in isolation is difficult, because code is about making ideas into actions. Code is a language that is about assumptions, and it's about driving machines to do what you want them to do, to make your own world. As a technologist, coming up with new ideas that are actually useful to people is almost impossible. It's just very, very difficult. People will tell you that is not true, that they have wonderful ideas: the problem is that everybody else has pretty much the same ideas, because tech builds on tech, the way that lanugage builds on language. English picks up bits and pieces of other words from other languages like a thief and technology advances upon itself to develop new things. So coming up with something that's genuinely new using a technological driver is really difficult, because the people who are already thinking about technology are not thinking about technology for what it might make [the humane], they're thinking about making more technology [more stuff].

Without a project goal, there's nothing to write. If you don't have a thing to make, you're not making a thing. You're arranging some ideas on a page, and that's all. And in isolation, that gets very lonely, because you do need to concentrate when you're coding, you need to concentrate for gigantic blocks of time - it's a creative practice that way. It's very hard to hold ideas in your head for a long period of time if you have to go off to meetings and do other things. So this is a creative practice that's about isolation that cannot be driven alone. HARD.

psXXYborg offered me an excellent opportunity to think about how new, expensive, "free" technologies might be used to make hard things easier. Technically hard things. The magic in code is that it disappears. You only see code when something goes wrong, because otherwise, it simply acts as though it is a glass panel. And that is mostly what technologists want code to do. We pretty much want it to retreat into the background, because when it's centre stage and when it's on display, people become frightened of it, sort of like how if you put a word problem in front of a 13-year-old, they screetch and

run to the other room. Just a problem. It's a thing that happens. So until something goes wrong, code is invisible.

The joy of fixing things is that when they go right, the code disappears again. So that's the whole idea about the invisibility of code. I'm working on that further for my thesis, because I think it's a big idea, and it's a good one to write down in the context of the arts.

There are lots of interesting threads in the disappearance of a language that drives the mechanics of your world. These are languages that build your medical devices, your cars, your phone: having them be invisible is an interesting problem the same way the underlying gendered assumptions in language is an interesting problem to overcome from the point of view of the legal system. It's an interesting problem. What I mean by an interesting problem is a problem that's worth spending a long time on, teasing out all the details. It's like a math problem that's made up of words. Not a word problem, a problem with the words. So, language assumes culture, Javascript assumes callbacks. That means you write it in reverse. It's sort of like being Ginger, you dance backwards in heels the whole time.

As a coder, what you write becomes real in a tangible sense, making interesting new things is difficult, thinking about what they might be is hardest, particularly alone - so, how do you fix this? You find a gang. A girl gang! The arts have an audience, but their audience is limited by technology [**LISA notes** If you have a particularly brilliant painter or a particularly brilliant graphic designer or an awesomely amazing musician, that is very fine, however, if they cannot record their information, or transmit it to a broader audience, then they will be broke. We know this is a problem. The next problem is how can we line up the technology so that they don't go broke after they do have the audience. We're still working on that one. It's a scary one.

Technology has a profit margin, but technology, because it builds on itself in a recursive manner, after education and privilege has given people the ability to write it, often has a very limited cultural outlook. People burn out in the tech field all the time. They burn out constantly. Google hires people for the express purpose of burning them out building machines. Because that's what we do. We write documents that are machines. They're laws. They're just laws made of a different kind of language. And they are breakable. There's an entire body of people who work breaking the laws of the language that we write into our documents. We call them hackers. They're very valuable. We need people who can break the laws to show us where they go wrong.

Together, good art and good technology can make good experiences. What I mean by good experiences is experiences that ask questions or display well-rounded characters

or use any of the many guidelines and theory that we have that drive the Humanities and the arts more broadly. If we pair people off, then we lose the part where the technologists are scared of the artists because the artists speak a mysterious language that the technologists do not understand, and the artists can learn that what looks like magic - a beautiful glass panel that just does what they say - actually takes a lot of work, it's not work that is unquantifiable either. It's built in code commits and checkins, you can see where the period goes, it isn't being a wizard. On either side. It's being a mechanic. It's assembling things well and putting them together and bolting them down and recording that so that it can be done again by another person. This is true on both sides of the equation. Feminism is equality. Art is technology.

By driving with art, rather than technology, the experience can be newer, and less expected. This is because of the aforementioned recursiveness of technology. Tech builds on tech, almost unquestioned, but art builds on culture and culture has a way of slipping around. When you are not looking at it, it moves in the dark. You think that teddy bear is sitting on your shelf but you wake up and it's at the end of your bed and let me tell you, you will scream. Culture can be seen as nothing but questions, questions like "How might a person be?" I think the answer is better.

Bibliography

Anthropy, A. (2012). *Rise of the videogame zinesters: how freaks, normals, amateurs, artists, dreamers, dropouts, queers, housewives, and people like you are taking back an art form* (Seven Stories Press 1st ed.). New York, USA: Seven Stories Press.

bbondy@mozilla.com. (2013, November 29). Contributing to the mozilla codebase. *Mozilla*. Retrieved from https://developer.mozilla.org/en/docs/Introduction

Bissell, T. (2013, September 25). Poison tree: an open letter to nico bellic about 'grand theft auto v'. *Grantland*. Retrieved from http://www.grantland.com/story//id/9719678/tom-bissell-writes-letter-niko-bellic-grand-theft-auto-v

Carver, J., C. Faber. (2014). Iv. Retrieved from https://github.com/jennie/iV

Cixous, H. (1976). Laugh of the medusa. *Signs: Journal of Women in Culture and Society*, *1*(4), 875. Retrieved from http://www.dwrl.utexas.edu/~davis/crs/e321/Cixous-Laugh.pdf

Ensmenger, N. (2010). Making programming masculine. In E. by Thomas J. Misa (Ed.), *Gender codes: why women are leaving computing* (pp. 115–141).

Faber, J. (2013, November 23). Iv. *GitHub*. Retrieved from https://github.com/jennie/iV

Galloway, A. (2006). *Gaming: essays on algorithmic culture (electronic mediations)*. Minneapolis: University of Minnesota Press.

Glanville, R. (2014, January). *Cybernetics*. Lecture at OCADu.

Klimas, C. (2009). Twine. Retrieved from http://www.twinery.org

Martin, B. (2012). Design charettes. In B. Martin (Ed.), *Universal methods of design : 100 ways to research complex problems, develop innovative ideas, and design effective solutions* (pp. 58–59). Beverly, Ma.

Norman, D. A. (2002). *The design of everyday things*. USA: Basic Books.

Pepi, M. (2013). Digital proletariat: the spectacle of the "internet" and labor's dispossession. Retrieved from http://www.thestraddler.com/201311/piece7.php

Plant, S. (1997). *Zeros + ones: digital women and technoculture*. London: Fourth Estate Ltd.