PAPER DOLLS IN CYBERSPACE: VISUAL HACKING AND REPRESENTATION IN THE TEACHING OF COMPUTER PROGRAMMING

A Thesis
submitted to the Faculty of the
Graduate School of Arts and Sciences
of Georgetown University
in partial fulfillment of the requirements for the
degree of
Master of Arts
in Communication, Culture, and Technology

By

Tracy Carlin, B.A.

Washington, DC
April 29, 2013
PAPER DOLLS IN CYBERSPACE: VISUAL HACKING AND REPRESENTATION IN THE TEACHING OF COMPUTER PROGRAMMING

Tracy A. Carlin, B.A.

Thesis Advisor: Garrison LeMasters, Ph.D.

ABSTRACT

Despite the importance of coding in an increasingly computer-dependent age, the Higher Education Research Institute found from 2000 to 2005 that the number of college freshmen listing computer science as their probable major dropped by 70% in the US. Meanwhile, the U.S. Bureau of Labor Statistics predicted in 2005 that 65% of job openings from 2004 to 2014 in science and engineering will be in information technology. Can the teaching of computer languages be improved? This thesis will explore two underused tools in the computer science classroom: digital representations, or avatars, and hacking, or the creative reworking and remixing of code. It examines how six adult graduate students approach hacking the same program, adding details and new forms of interactivity to reflect their own interests and passions. It also explains how the misunderstood concept of hacking code could work as a useful educational tool for an active learning environment, in the style of Seymour Papert’s Logo learning language.
# Table of Contents

Introduction ................................................................................................................. 1

The State of Computer Science Learning Today ....................................................... 3

Not Enough Computational Literacy to Meet Contemporary Economic Demand .... 3

The Ideal Classroom – Learning without Curriculum, Turtle Programming, and the New Culture of Learning ................................................................. 6

Papert and Piaget ........................................................................................................ 6

Douglas Thomas, John Seely Brown and the New Culture of Learning ................. 11

Case Studies: Negative and Positive Examples of Contemporary Programming Language Learning .................................................................................................. 14

Three Contemporary Case Studies: An Examination of Current Models............. 14

Teaching to the Test: The Inflexible High School Model .................................... 14

Computer Programming 101: How the Introductory College Experience Improves a Still-Failing Model ..................................................................................... 18

Semi New Culture – A Cyberspace Seminar ......................................................... 20

Positive Outliers: Online Learning, Video Games, and Representation through Avatars ................................................................. 27

The Benefit of Avatars and Representation ......................................................... 32

The Proteus Effect ..................................................................................................... 32

The Why of Whyville ............................................................................................... 33

The Paper Doll Project ............................................................................................ 36

Methods and Structure ........................................................................................... 36

Processing and the Coding of the Aesthetic ......................................................... 39
INTRODUCTION

This is not a project about computers. This project is fundamentally about people. In the past one hundred years the systems of communication have fundamentally changed. What was once done through paper is now done over a computer, what once took weeks to transport across the country takes seconds over email. The most powerful computers that once took up an entire room can now fit easily into a single computer chip. This makes code, the language of computers, incredibly powerful. Digital literacy is essential for 21st century communication, whether it’s as a student writing a paper, a worker drafting a proposal, or an entrepreneur selling their goods over the Internet. Society has come depend on computers. Yet like many science, technology, engineering and math (or STEM) disciplines, there is a shortage of Americans with sufficient education in computing to meet the demand in this global economy (Casey, 2012, 1).

This project is about people and their relationship with computers. The following pages will look at ways to make computer literacy and coding a more widely-learned enterprise. Specifically, this project will examine how hacking, or manipulating existing code to create a new program or application, can be used in the computer sciences for a more accessible teaching environment.

This project comes in three parts. The first reflects on the state of computer science today and investigates the possible improvement of student engagement through the use of avatars, or digital forms of self-representation. The second part describes the process of creating the “Paper Doll Project,” an application which uses an avatar to explain simple code concepts in object-oriented programming. The third part introduces this code to several students in the graduate-
level Computational Expression class and follows how they hack the original avatar to match their personal tastes and interests. The goal of the project is to see if avatars, a visual representation which can be customized at the individual level, might be a more accessible introduction to important computer science concepts than less hands-on, less creative approaches.
THE STATE OF COMPUTER SCIENCE LEARNING TODAY

Not Enough Computational Literacy to Meet Contemporary Economic Demand

In an increasingly digitized society, the ability to code creatively is a highly sought skill, one that is essential to survival in a computer-mediated world. Those that can build their own software have an advantage over those without such talents. Unfortunately, the United States is lagging when it comes to digital literacy and computer programming. There are iPads and computers in many schools, but the skills required to create applications are not being taught to the extent required in the Information Age.

Many American students are not reaching even basic levels of computer science literacy at the secondary level, which puts them at a severe disadvantage when they attempt to pursue a computer science major in college or a computer-related occupation. The Higher Education Research Institute announced recently that while US high schools are making strides in teaching students to code, the same schools are not adequately preparing students for advanced computer science education (Pryor, 2006, 16). The federal government recommends that students receive at least one semester of computer science in high school, but implementation varies (ibid). In total, 61.6% of students take the recommended semester, well below the number of students that reach the basic requirements of other subjects (ibid.). The problem, according to the report, is that “schools are having difficulty providing a half-year of computer science to large numbers of high school students, many of them bound for studies at four-year colleges” (ibid.). In particular, the study found women less prepared than their male peers (56.9 percent versus 67.2 percent) (Pryor, 2006, 17). This is despite the fact that the use of personal computing has risen considerably for women and girls, from 23.5 percent in 1985 to 85.8 percent in 2005 (Pryor,
2006, 27). Even though computers are becoming essential within a 21st century learning environment, the skills to develop software are not a priority for the American educational system.

The lack of computer science education in the United States has led to an increased demand for programmers and a shortage of talent to fill those positions. The Bureau of Labor suggests, for instance, that computer and mathematical occupations are expected to grow by over twenty percent over the next ten years (Casey, 2012, 2). This is a significant number of new jobs in a relatively stagnant economy. The increased demand for code-literate individuals could also have an impact on the rate of unemployment for American students. Recent studies from the Joint Economic Committee indicate that one in five young workers is unemployed (Maloney, 2010). In comparison, the Bureau of Labor study found that for those in computer and mathematical occupations, the unemployment rate was 4.1 percent – less than half of the overall unemployment rate of 8.9 percent (Casey, 2012, 3). The increased value and relative permanence of computer-related positions should make programming and computer science education far more of a priority than it is treated as in United States high schools and universities.

Women in particular are abandoning computer science education. While the number of women earning bachelor’s degrees has risen three percent since 1999, the number of women majoring in computer science declined by ten percent (Casey, 2012, 5). Similar statistics from community colleges show women earning associate’s degrees in computer science have dropped from 42% of degree recipients to 25% (NSF, 2012). The most recent numbers from the Computer Research Association reinforce evidence of this trend, the population of computer
science undergraduate women dropping from “13.8 percent in 2009-10 to 11.7 percent in 2010-11” (CRA, 2011). These decreases continue despite the fact that through 2011, the number of computer science majors has slowly and steadily been growing. In the workforce, only 27 percent of individuals working in computer science and math positions are female (Casey, 2012, 5). These trends contribute to the overall dearth of code-literate professionals, and correcting the gender gap is an important part of fixing the problems with computer science learning overall.

Current in-place learning practices have not resulted in increased supply for the United States’ economic demand. Instead, students are ill-prepared and alienated. Without a new approach, there is no indication that the poor numbers will improve. The following sections will take a closer look at educational approaches, both theoretical and practical, which may help (or hurt) the pressing need to improve computer science pedagogy.
Papert and Piaget

The basis for many of the elements of this project is educational theorist Seymour Papert. Papert’s main contributions to the field include theorizing the application of his teacher Jean Piaget’s ideas about an “active learning environment” to STEM education. Piaget is a key figure in the development of the constructivist learning theory. Piaget, a scholar of childhood cognitive development, writes that learning is “a process of adjustment to environmental influences.” (Pritchard, 2009, loc. 576). Learning happens when “the mind of the student mediates input from the outside world to determine what the student will learn” (Pritchard, 2009, loc. 601). A more passive form of learning, such as a lecture, will not be as effective if a student is not actively engaged. Assimilating previous experience with new knowledge in an active way is key to productive learning under this philosophy. Papert's insight is to take Piaget's generalized theory of learning and apply it to improving STEM education, especially computer science.

Papert used active learning environments to teach young children computer programming as far back as the sixties. He wrote in his book *Mindstorms* that to learn to program, a learner needs to master “procedural thinking” (Papert, 1993, 135). Procedural thinking, which I will also refer to as procedural literacy, is defined as the capacity to think like a computer. Procedural thinkers can understand computational cause and effect, following how one line of code affects what came before, and how to manipulate each line to desired effect. Procedural thinkers can also understand iteration, or going over the same series of steps multiple times, to see how minute changes in component parts can give rise to change in the whole. By teaching children
procedural literacy, Papert asserts that “children could also benefit…from the way in which computer models seemed able to give concrete form to areas of knowledge that had previously appeared so intangible and abstract” (Papert, 1993, 23). In this way Papert extends Piaget’s ideas of active learning to teach computer literacy. His result is the Logo computer language, a programming primer language specifically designed to teach young people these skills with a minimum of abstraction.

Papert’s Logo language uses an animated turtle, which follows cues from the programmer to draw increasingly complex shapes and figures. Papert designed Logo to be as easy for children to understand as any other natural language. As he says in Mindstorms, "It is possible to design computers so that learning to communicate with them can be a natural process, more like learning French by living in France than like trying to learn it through the unnatural process of American foreign language instruction in classrooms" (Papert, 1993, 6).

To Papert, part of the unnatural nature of teaching programming is due to its reliance on abstraction. While programs are no longer stored on punch cards and no longer require days to execute (as they did in the early days of computers), many classrooms continue to approach teaching how to program in way that abstracts code from the experience of the average learner, and therefore is not ideal for active learning. Logo corrects this abstraction.

For an example, let’s look at a sample of Logo code.
Figure 1 displays the simplicity and tangibility of Logo. The image presents a simple function in Logo which draws a chair. The “turtle” is represented by the triangular figure at the top. On the right is the code. Logo boasts simple words and an uncomplicated grammar, comparable to that of a child. The turtle turns and moves forward in order to draw the simple shape; the language initiates the turns and the movements with functions like “right” and “forward.” The numbers indicate the length of the lines to be drawn. The language's immediacy is intentional: Young programmers find the meaning of the code intuitive, and when they are confused they can "act out" the code in the real world. Students leave their desks in order to play the part of the turtle itself, moving forward and turning on command. In this way Logo, is an active learning simulation with a minimum of abstraction, and is an ideal environment in which to acclimate young coders to computational thinking.

For the sake of comparison, here is a sample of FORTRAN (or the IBM Mathematical Formula Translating System), a common programming language from the late 1950’s and 60’s:
PROGRAM MAIN
  INTEGER N, X
  EXTERNAL SUB1
  COMMON /GLOBALS/ N
  X = 0
  PRINT *, 'Enter number of repeats'
  READ (*,*) N
  CALL SUB1(X, SUB1)
END

SUBROUTINE SUB1(X, DUMSUB)
  INTEGER N, X
  EXTERNAL DUMSUB
  COMMON /GLOBALS/ N
  IF(X .LT. N) THEN
    X = X + 1
    PRINT *, 'x = ', X
    CALL DUMSUB(X, DUMSUB)
  END IF
END IF
END (Miller, 2002)

Unlike Logo, these lines of FORTRAN are not intuitively comprehensible. While there are terms in FORTRAN that have a basis in human language – PRINT, CALL, INTEGER, END – it is unclear at first glance what each of these commands tell the computer to do. This language would be difficult for adults to learn, let alone children and teens new to computer science. The abstract nature of FORTRAN would frustrate an average learner. In short, FORTRAN is not an ideal active learning environment, as Logo is.

Papert felt strongly about the constructivist approach of matching learning environment to experience; one part of his approach to teaching programming is helping students equate coding with experiences in their daily life. To Papert, children, "like other builders...appropriate to their own use materials they find about them, most saliently the models and metaphors suggested by the surrounding culture" (Papert, 1993, 19). Papert sees teaching children to
program as a way for them “to do something” for their own reasons, not just because it is required, and through this action turns programming into “a source of power...experienced as such from the moment it begins to form in the child's mind” (Papert, 1993, 21). This type of naturalized language learning environment is an example of what Piaget calls “teaching without a curriculum.” Teaching without a curriculum is not just “supporting children as they build their own intellectual structures with materials drawn from the surrounding culture,” as Papert defines the term, but also designing the classroom so that learners tackle their subjects with the desire to find their own power in the utilization of these skills and knowledge (Papert, 1993, 32-33). This stands in stark contrast to the current model of STEM and computer science training that uses a model of knowledge “taught and taken as medicine” (Papert, 1993, 48). Understood in the context of the current shortage of programming literacy in the United States, Papert’s success with teaching children Logo suggests that teaching within the learner’s experience, and thereby showing her the power of the knowledge in question, might be preferable to the more common “teaching as medicine” approach.

Papert used Logo to revolutionize STEM and computer language pedagogy by giving children a tool they could manipulate on their own. When a child is “Turtle programming,” the learning environment is one where they can make mistakes repeatedly without serious ramifications. Over time they learn to take control of the Turtle for their own creative purposes. He describes the experience of using Logo and Turtle programming as both exciting and highly active:

Learning to control the Turtle is like learning to speak a language. It mobilizes the child's expertise and pleasure in speaking. Since it is like being in command, it
mobilizes the child's expertise and pleasure in commanding. To make the Turtle trace a square you walk in a square yourself and describe what you are doing in TURTLE TALK. And so, working with the Turtle mobilizes the child's expertise and pleasure in motion” (Papert, 1993, 58)

Instead of the more dissociative experience common in many computer science curricula, Logo teaches programming in a highly concrete and applicable way. It is understandable enough that its real-world effect is immediate, and is easily adapted to the personal experience of the learner. These elements result in a class full of computer literate and highly-motivated students who know how to use the computer for their own creative purposes. The contemporary American classroom could learn a great deal from Logo.

The inclusion of the turtle avatar within the interface is key. Logo is not just a language that can draw a flower; the turtle draws the flower for them. Papert explains in Mindstorms why he chose to include the turtle within the interface: the tiny avatar allows children to connect to the abstract world of computer programming via an amiable metaphor (Papert, 1993, 65). This aspect too could be beneficial to the modern student.

Douglas Thomas, John Seely Brown and the New Culture of Learning

Besides constructivism and active learning environments, new educational theories have also suggested how modern 21st century tools have changed the way people learn. John Seely Brown and Douglas Thomas call this new educational experience a “new culture of learning” (Thomas, 2011, 17). In a new culture of learning, education is not delivered in the classroom but via the Web with tools like YouTube, Facebook, and online games. This learning is purely voluntary and takes place in a playful atmosphere. Thomas and Seely Brown define this new culture of play and learning in this way:
Play can be defined as the tension between the rules of the game and the freedom to act within these rules. But when play happens within a medium for learning -- much like a culture in a petri dish -- it creates a context in which information, ideas, and passions grow. (Thomas, 2011, 18)

Because the new culture of learning does not have the structure and supervision of a traditional teacher, more than creativity and imagination are required to keep the learner engaged. To Thomas and Brown, a communal element is absolutely key to this engagement. They use the Massachusetts Institute of Technology’s educational software Scratch as an example (Thomas, 2011, 20). Like Logo, Scratch is designed to teach young children how to code (ibid). Scratch can not only draw figures, it also teaches children more complex code such as how to program games. Its true power lies in its online community where children can share their creations and, if they wish, modify them from the comfort of their own computers (Thomas, 2011, 21). The inclusion of these collaborative elements is what makes Scratch so different from a traditional classroom environment.

The new culture of learning is in many ways defined by this type of peer-to-peer learning. In this environment, whether it be on- or offline, learners lean on each other for breakthroughs, collaborating and forming new ideas on the fly outside of traditional educational structures. Brown defines peer-to-peer learning as similar to a college freshman’s experiences:

When students set foot on campus in their freshman year, they begin a learning experience that is governed only in part by their classroom interactions. Assuming they live on campus, sleep eight hours a night, and attend classes three hours a day, students are immersed in a learning environment for an additional thirteen hours a day. Simply by being among the people around them -- in study groups, for instance -- students are learning from this environment, participating in an experience rich in resources of deep encounters. (Thomas, 2011, 51).
This type of peer-to-peer interaction is fairly rare, even discouraged in a traditional classroom. In many cases collaboration is treated as cheating and the participants are punished. To Thomas and Brown an ideal learning program should not separate students into units. This separation isolates the learners from the rich peer-to-peer environment that a new culture of learning stimulates.

The new culture of learning could be applied to computer science education with excellent results. Professional programmers already utilize the peer-to-peer education system; even the most seasoned programmers will occasionally find problems outside of their experience and need to call on others’ expertise to resolve the situation. To treat each program like an examination – where students work independently, with only one chance to find the correct answer, and with no opportunity to build off of the work of others – ignores the way computer programming works in the real world. By combining the tools of Thomas and Seely Brown’s new culture of learning with the active learning environment of Papert’s Logo language, a more natural learning environment for budding programmers could be created.
CASE STUDIES: NEGATIVE AND POSITIVE EXAMPLES OF CONTEMPORARY PROGRAMMING LANGUAGE LEARNING

Three Contemporary Case Studies: An Examination of Current Models

In this section we will discuss three modern examples of how computer science is studied and explain why these case studies showcase an ineffective learning model. In the traditional classroom, represented by Syllabus A and B, there is a problematic focus on test-based assessment and active discouragement of peer-to-peer learning models. In Syllabus C, or the independent online model, the ideas represented by the new culture of learning are embraced, but its automated, teacher-less experience threatens to lead to a frustrating user experience divorced from creative coding. While the models gradually improve, none of these examples meets the standards for a constructivist, active learning environment — an environment that may be ideal for learning how to program. These syllabi were freely found on the Internet and any identifying information has been anonymized in Appendix A. They are not the only models available, but they are examples of a typical Computer Science classroom environment.

Teaching to the Test: The Inflexible High School Model

Syllabus A belongs to a high school Advanced Placement Computer Science class. Advanced Placement courses are high-school level but can in certain cases apply to a college transcript. These classes are usually only available to junior and senior high school students, although there are exceptions. The syllabus indicates that the class is open only to students with previous computer science experience. They also must have good grades from previous honors math courses and a PSAT score of 58 as a “positive indication of readiness” (See Appendix A, Syllabus A). The requirements exclude implicitly those who do not consider themselves
mathematical thinkers. Those that have not tested well on the PSAT are further removed from consideration. Here, the student in question must have years of preparation in math before she can be “allowed” to code. Papert demonstrated over fifty years ago that advanced mathematics are not a necessary or useful pre-requisite to programming, and yet this contemporary classroom still purposely excludes the “math-phobic.”

The class structures itself around what John Seely Brown and Douglas Thomas would call a “mechanistic” approach (Thomas, 2011, 34). Thomas and Brown define mechanistic learning as a situation where “learning is treated as a series of steps to be mastered, as if students were being taught how to operate a machine or even, in some cases, as if the students themselves were machines being programmed to accomplish tasks” (Brown, 2011, 35). It is directly comparable to Papert’s “teaching as medicine” concept. In this case, the teacher is programming students to get an acceptable test score on the AP Computer Science exam, the test which decides if the students can count their high school experience to their college transcripts. It is in the teacher’s best interests to program students to answer questions rather than prepare students to code on their own. To that extent, any programming actually accomplished during the year-long class is simply a byproduct through which the students have learned how to get a better score on an exam. This system removes any benefits students might gain through crafting their own programs and applications: In the end, they cannot develop the sense of power Papert envisioned.

Each unit of the course is paired with several chapters of readings; the syllabus indicates that the first three units are purely concept oriented. It is likely that this "disassociational" model
will lose students better suited to an active learning environment, boring them, alienating them or, even worse, making them feel incapable of succeeding before they reach the point of concrete application. Indeed, two full months pass before the first “programming unit” begins. A more constructivist approach would incorporate more uses of code earlier, to better acclimate computer science to the experience of the learner.

The main form of student assessment in Syllabus A comes in the form of regular testing, which prioritizes memorization over practical application. Since Java-specific programming and syntax is only taught beginning with the fourth unit, students must pass several units of abstraction before they reach the point of reading actual code. Even then there is no mention of student programs; homework comes in the form of chapters from a textbook. The focus on abstract reading and test-based assessment forms a pedagogy that prepares a student for answering questions about programming, instead of learning how to use Java to build and create programs.

Only in December, over three months after the class has begun, does the class shift to a focus on programming exercises. The assignment, “design and create a significant software project,” is suitably vague to allow any and all sorts of creative interpretation, and the amount of time given is significant – five months (Appendix A, Syllabus A). Unfortunately, the class structure takes twelve units to reach a practical, code-focused curriculum. Students who will thrive with a hands-on learning experience, as opposed to constant reading and test-taking, will have already become alienated from programming. A constructivist approach would try to
introduce practical coding assignments throughout the semester, or at least allow students to begin coding earlier than halfway through their academic year.

The “materials needed” section does not mention any kind of outside work with computers, meaning most of the homework is reading or writing, not coding. Students do need a USB drive for storing programs, but they also need dry erase markers and 3 x 5 cards. The 3 x 5 cards in particular suggest the lack of hands-on, independent, creative exploration in this classroom; cards are required because, in the words of the syllabus, “each student will, over the course of the year, create a set of flash cards to use as a study aid.” This kind of assignment indicates a focus on memorization, a requirement that, to a significant extent, the modern programmer does not actually need: Since a programmer programs on a computer, they can always look up what they do not know. But this classroom relies heavily on memorization because the goal is not an adept programmer, but an adept test-taker. The ideal student in this class will be able to rattle off names, dates, and vocabulary for the AP Computer Science exam, but they will be left unprepared to use programming for creative application.

The behavioral expectations are also contrary both to Papert’s model of active learning and those of the new culture of learning defined by Thomas and Seely Brown. In this class, for example, students must remain seated at all times and stay silent when the teacher or a guest speaker is addressing the class; this limits the student’s ability to collaborate with each other or try and use “turtle programming” to solve bugs in a hands-on way. Insofar as students must commit to “passing the AP Exam,” the test itself becomes a higher priority than learning to build functional code. Students cannot touch equipment without express permission, meaning they are
likely not encouraged to work outside of class. During class, they are forbidden from using their time for any reason other than “learning the subject” of that class day. Recall that it takes two months before students get to the point within this syllabus to code. The students cannot collaborate, they cannot pursue goals that interest them, and they cannot create code for their own uses. This classroom environment will not produce enthusiastic, creative coders.

Syllabus A is a perfect example of a classroom modeled on teaching to a test. As Papert would characterize it, programming is here a medicine given to students to reach a sufficient grade on the Advanced Placement Computer Science test. The requirements of the course actively discourage anyone who lacks honors-level math experience, cutting off math-phobic students. For those in the class, creative expression comes only after months of test-based assessment. The result are students capable of taking a test. But the United States does not need test-takers in the professional world; it needs creative programmers.

Computer Programming 101: How the Introductory College Experience Improves a Still-Failing Model

Syllabus B comes from an introductory college programming course. Unlike the high school class, this course does not require any previous mathematical coursework or STEM specialization. Students from all disciplines are invited to enroll without any prior assessment or special permission. Because this is a college course, there is some expectation that the student would have passed high school mathematics. The students, by their very admittance to the university, have already completed a significant amount of prior education. However, these learners may still perceive themselves as poor at math. The open invitation therefore encourages the math-phobic to give programming a try.
This syllabus emphasizes that programming is a craft; as such, it seems to incorporate some aspects of active learning directly into the curriculum. The professor wants his students to feel free to program no matter their discipline or preferences. He praises active and hands-on learning. The ideal participant in this class is described as “paying close attention, raising questions, making suggestions, posing challenges, casting doubt, sharing experience, and so on – basically, any means that demonstrates their interest and enthusiasm.” Demonstrated interest and enthusiasm are usually the hallmark signs of an active learning environment. Clearly this professor understands Papert. However, the vast majority of class time is spent on discussion and assessment over practice. What is more, the syllabus explicitly states that programming assignments are to be done alone. This eliminates the possible establishment of any useful collaborative environments and strips out the benefits of the new culture of learning. This course, while more active than the course described in Syllabus A, still has not accepted some key concepts in the new culture of learning such as peer-to-peer learning.

This syllabus, like Syllabus A, also ignores practical programming assignments in favor of more traditional test-based assessment. Students are responsible for far more readings than they are examples of computer code. The final assessment is a test, which means that students will prioritize memorization, which has little use when called upon to code in a professional environment. At the end of this semester the student will understand vocabulary and syntax, but they will have less experience exercising their creativity or working with concrete assignments. Because this is an intro class, its structure may alienate students who might otherwise appreciate more creative and concrete ways of learning how to code. This is a serious concern when a third of math, physics and computer science majors eventually change majors, leaving STEM
disciplines entirely (NSF, 2012). With so many creative and intelligent students giving up on
computer science ambitions, introductory courses like this ought to attract rather than alienate
inexperienced computer programmers. This shift could be accomplished with a minimum of
effort: Consider how this course could easily be converted to present a more active curriculum.
The professor could prioritize the more hands-on, concrete experience of coding, or tie each
reading assignment directly with a programming challenge, culminating in a final project that
makes students display their understanding of programming concepts by developing a program
that suits their interests.

**Semi New Culture – A Cyberspace Seminar**

Syllabus C comes from the popular online learning environment, Code Academy
(www.codecademy.com). Code Academy is substantially different environment than a traditional
classroom. The web-based learning environment skyrocketed in popularity after Mayor Michael
Bloomberg declared that his 2012 New Year’s Resolution was to “learn to code” using Code
Academy (Glazer, 2012). The tweeted declaration led to thousands of new subscribers. Instead of
working with a human teacher, the user plugs their code into a text window on the website; this
interface then reads and executes the code. Then it determines if the student has learned the
lesson. If they have correctly written the code in a way acceptable to the automated check, then
the student can move on to the next lesson. Code Academy has incorporated some elements of
active learning and the new culture of learning into its structure, but the fully online, automated
pedagogy can be overly restrictive and frustrating for budding programmers.
Code Academy, unlike the advanced placement high school or introductory college course, is open to anyone with access to the Internet. The website does not discourage the math-phobic from trying to code or force learners to reach a required level of math proficiency to enter. The website also avoids required readings, tests, or other memorization-based assessments for the programming concepts it teaches. The only requirement to progress through Code Academy is demonstrated proof that the student understands the concept of the unit open before them. This competency is always demonstrated with code. Once the basic request is fulfilled, the student is moved on to the next component. This is a more open, active, and constructive approach than the one used in the traditional classroom environments previously discussed.

Code Academy has added more peer-to-peer elements to its website through its “groups” system (http://www.codecademy.com/groups). The website supports the use of “groups” to help learners connect over common interests or goals. There are those grouped around languages, such as JavaScript, proficiency, and even ones based around career networking, profession, and geographic location. In addition, a student can create private groups exclusively for friends or work colleagues. The website keeps track of and scores both group activity on the forums and the coding activity of the group as a whole, in a sense gamifying peer-to-peer learning. The groups function is fairly new as of April 2013, and little has been written about its effectiveness. However, based on other case studies of peer-to-peer learning, the group system should lead to more collaboration and enthusiasm.

But these advantages exist with some drawbacks. Within the lessons themselves, for example, there is no opportunity for group collaboration or even traditional teacher-student
interaction. The lack of human instruction and the focus on automated response means that Code Academy must rely on an overly restrictive means of assessing its students, which can lead to poor instruction and frustration. The following examples, therefore, will explore these problems by examining Code Academy’s Introduction to JavaScript syllabus. JavaScript is a common programming language used to script elements in a diverse range of applications, most commonly on web pages. It can run within every Internet browser and is therefore a useful tool for those putting together a complex website. Students interested in web development could make good use of the lessons contained in this syllabus. However, as I will demonstrate, the Code Academy environment does not sufficiently prepare a learner for taking these lessons and using them to create custom code.

Figure 2:

One of the first lessons, as seen in Figure Two, provides an example of how the automated nature of Code Academy restricts its learners. This section teaches students how to
code mathematical functions into JavaScript by typing in an equation. JavaScript can understand and calculate any basic equation by simply typing the numbers in and pressing Enter for a response. In this case, the website asks for “any addition operation.” If the user enters a multiplication operation, demonstrating their understanding of the subject (encoding mathematical functions generally) but ignoring the explicit instructions of the website, the Code Academy interface tells the learner they are wrong. This, despite the fact that the JavaScript window still accepts the equation and calculates the answer. If the lesson simply asked for a mathematical equation, without specifying the type, the website could both allow creativity and check the students’ understanding of the lesson. As it stands, however, the student is faced with a confusing condition where her response is both right and wrong.

Figure 3:
Figure 3 is an example of Code Academy teaching the user one aspect of JavaScript without giving her the skills to work independently. In this case, the website misses the opportunity to teach the importance of quotation marks.

For this lesson the website requests that the user write code using the JavaScript “confirm” function. This piece of code allows for a website or a piece of software to create tiny pop-up windows on which the user must (“confirm”) to dismiss. In many cases, such as in Microsoft Word, a window like this (a "modal" window) is used to ask a user if she wants to close a document without saving it. Websites may use similar code to prevent a user from leaving the webpage without first reading certain information. In short, the function forces the user to actively agree to some condition.

Because this type of code includes text data, otherwise known as a “string,” quotation marks are used to differentiate language written in human terms (in this example, the word “Puppies”) and language written in computer terms (the confirm() function). It is an essential grammatical requirement to use strings. In Code Academy, the user adds one quotation mark and starts to type her string, the second quotation mark is automatically added at the end without informing the user. An experienced user aware of the quotation mark requirement can accidentally create an error when she tries to add the second quote mark, while a neophyte coder will take for granted Code Academy’s automated process.

A better approach would be to let the student fill in both sets of quotation marks themselves. This method will reinforce the importance of using quotation marks when incorporating strings of text, and the user will avoid unnecessary errors that do not help the
learning process. Continuing to use the automatic quotation mark “feature” will otherwise cause problems in the future. An untaught coder who wants to program their own JavaScript functions with strings may only type the initial quotation mark. If so, the program will be filled with errors, and the user will be frustrated. They may eventually recognize that the second set of quotation marks are missing, but this epiphany is not guaranteed.

Importantly, Code Academy is more open and active than the traditional classroom environment; however, this does not translate into the ability to code creatively. When the learner passes the point of first encountering basic concepts, Code Academy claims its students can now build their own applications. The syllabus for JavaScript, for example, lets students build games such as Rock, Paper, Scissors and Blackjack once they have “graduated” from the introductory concepts. This is their attempt to translate Papert’s ideas of treating learning environments as a way to claim power. But even in these “advanced” lessons the automated Code Academy interface limits its students.

Figure 4:
In Figure 4, Code Academy rejects the users’ code due to a difference in capital letters. The interface wants a variable named “userChoice” and the user inputs “UserChoice.” Variables are simply locations in code that may or may not hold a value; capitalization means nothing. The only requirement is that a variable is “declared.” To declare a variable in JavaScript the user must say that there is a variable (var) and that it has a name (in this case, UserChoice). As with the previous example of the arithmetic operation, JavaScript understands what the website does not. The program runs perfectly even as the interface warns of an error. The hint located below the code will only confuse the user, as it claims the necessary variable has not been declared. A user who does not notice the minor capitalization error will be frustrated with her inability to continue. A minor error could lead the user to quit or lose interest in continuing her education.

Code Academy is a helpful system for teaching programming, far more active than the coursework described in the high school and college classrooms. Its efforts to include more peer-to-peer learning elements shows an interest in adapting to the new culture of learning. However, its automated nature leads to bugs that frustrate the user and may actually prevent those capable of programming from moving forward. Because of the stringency of the guidelines, it is difficult to imagine how a system like this one might be made to understand a student’s individual creativity, even though the website markets these courses as a form of agency. Code Academy is a step forward, but the system can be improved for a more active, collaborative learning environment.
Positive Outliers: Online Learning, Video Games, and Representation through Avatars

The following section describes positive outliers in STEM and computer science education. The following examples, contrary to the case studies explored earlier, demonstrate that a passion for computational thinking can in fact be imbued in students, including women and girls. Significantly, these cases are outside of traditional learning environments. In fact, writer James Paul Gee describes STEM passions forming among girls exposed to the unique learning environment of video game fandom.

James Paul Gee has written extensively on active learning environment and learning styles in the 21st century. In particular, he has focused on video games as rich learning environments. He argues that video games, their online fandoms and communities, and MMORPGs can be used to help uncover fundamental tools for other educational technologies or new learning classrooms (Gee, 2007). In 2011, he released a book on Maxis’ *The Sims* (see Figure 5 for a screenshot). *The Sims* is one of the few video games wherein researchers have confirmed a strong female fan base. It is, by its very nature, a game about avatars and representation.

An avatar is a digital representation of a person within a synthetic world. The relationship between a user and their avatar has become a subject of study recently, as more people interact using avatars online. In the article “On the Relationship between My Avatar and Myself,” an avatar is defined in this way:

Avatars mediate participants' interactions with each other and the environment. Avatars sit, walk, run, drive, fly, teleport, dance, and communicate with other avatars. The avatar is the “face” (and body) of a person within the virtual world.
People can freely express themselves in the appearance of their avatars, limited only by their imaginations and the available technology. (Messinger, 2008, 2, 14) Messinger offers a number of explanations about why avatars are important to study: they can enhance the users’ sense of self-worth; they can make their owners behave in more extroverted and confident manner (Messinger, 2008, 2, 14). Importantly, *The Sims* is different from the synthetic worlds Messinger studied because the game itself is single-player, confined to one computer. However, an online community of fans has emerged, and this community has developed a rich source of peer-to-peer learning around customizing and improving *Sim* avatars, colloquially called Sims.

*The Sims* gameplay is deceptively simple: players build small, simulated humans (the titular Sims) and then unleash them into a simulated community, where they live in homes the player designs and work at simple tasks as the user sees fit. For Gee, *The Sims* is an unexpected but excellent virtual environment for gender-neutral STEM learning: the learning that *The Sims* engenders is not focused on specific disciplines (where gender rapidly becomes an issue) but forms instead based on the personal passions of the investigator. Gee shows how those interested in design, those interested in creative writing, even those interested in social justice flocked to *The Sims* as a means for learning and expression. The following cases, as documented by Gee, illustrate among players a desire to improve or further customize their Sims and the environment that these Sims “live” in. This desire translates into enthusiastic, passionate learning directly connected to the personal experiences of the student.

The first example is a traditional student, Jade, a participant in the after-school Tech Savvy Girls program. Gee argues that through her creative use of *The Sims* and *Second Life*,
another virtual environment focused on avatar customization and creative environments, Jade went from a computer novice to an expert (see Figure 6 for an image of Second Life). The teenager grew into a technical pro, chiefly by replicating real-world clothes digitally and transferring them into The Sims. To modify her Sims, Jade learned a number of computer skills, including professional design software like Adobe Photoshop. She also learned to "mod," or modify, her copy of The Sims for her own uses, “the very sort of practice that many consider a key gateway for the development of high-tech skills among boys” (Gee, 2010, 63). Within a relatively short time, Jade went from using computers purely for social networking — such as keeping up with friends via Myspace — to tackling more technical, design-based challenges.

What is interesting about Jade’s story is that she was not particularly interested in playing The Sims; in fact, of the girls in the afterschool program “[Jade] was the least enthusiastic of the girls about The Sims... the game no longer challenged or excited her” (Gee, 2011, 64). It was only when the club’s leaders encouraged her to experiment with designing Sim clothing that she became interested in both The Sims as a game and, more significantly, in the technical apparatus of the software behind the game. She grew even more invested when she discovered a mod that let her design clothes for plus-size Sims, which better reflected her personal appearance (Gee, 2011, 66). Through Gee's presentation of the research, it becomes apparent that the avatar-like Sims, with whom Jade identified via her customization of the experience, fueled a passion for computing that her school could not.

Because Jade’s mentors had never designed clothing for Sims, she relied less on the traditional classroom (as represented by the club) and more on Internet tutorials and online
communities; her increasing proficiency impressed her colleagues in the club sufficiently that “eventually it would be Jade and her success, not Betty and Beth [the club’s adult mentors], who motivated the other girls” to learn how to customize their Sims (Gee, 2011, 68). Jade is a successful example of peer-to-peer learning in an online environment and in a traditional classroom. Through the same kinds of interactions that Seely Brown and Thomas laud in their new culture of learning, Jade thrived in a STEM-rich environment.

Jade came to see computers, in her words, as “a source of power.” The Sims helped her develop an enthusiasm for computational occupations sorely needed in the current United States economy. Tellingly, however, despite this enthusiasm and progress, Jade's school eventually blocked her from accessing their computer science and programming classes because of poor math grades (Gee, 2011, 71). Her case warns against the myopia of traditional schooling (which “failed to acknowledge, recruit, or enhance her new passion for design and computing”) even as it illustrates how virtual representations and avatars can inflame a passion for technical skills when these technical skills are required for proficiency (Gee, 2011, 82). Where the school failed her, The Sims online community continued to challenge and encourage Jade, and she continued to advance her skills.

Jade represents both the limitations of traditional classrooms and the power of peer-to-peer online learning. However, the new culture of learning is not limited to traditional students. Online peer-to-peer learning has no barriers based on age, and older learners can take just as much advantage as a teenager. Gee discusses the example of Tabby Lou, a popular designer of Sims custom content (Gee, 2011, 84). For Gee, Tabby Lou represents the success of the “new”
culture for non-traditional students. Unlike Jade, Tabby Lou did not have a community like the after-school Tech Savvy Girls program to introduce her design; in fact, health issues prevent Tabby Lou from engaging in traditional learning environments (Gee, 2011, 85). Nevertheless, using the same tools that Jade did (including the online communities and tutorials), Tabby Lou became an internationally-recognized custom-content designer for *The Sims*. Gee describes her transformation:

Tabby Lou could have viewed herself just as a retired, shut-in grandmother. Instead, she came to view herself also as a designer (for a video game), as a mentor and helper, and as an international contributor. This is not what we typically associate with retired, shut-in grandmothers. The world has changed. Like Jade, Tabby Lou is typically untypical.

Tabby Lou is an example of an independent learner teaching herself computational skills to improve a synthetic experience. This is accomplished in part by an oblique relationship to the game itself. Initially Tabby Lou began designing *Sim* products because her young granddaughter wanted a purple toilet for her Sims’ house (Gee, 2011, 85). Tabby Lou, like Jade, did not enjoy *Sims* game play nearly as much as she did designing custom content. In her words, “I enjoy being a creator more than being a gamer” (Gee, 2011, 90). For Tabby Lou, the Sims themselves served as a gateway to the more interesting and satisfying world of modification and design.

Another older woman, fifty-three year old Izaru, “mastered Photo Studio, Photoshop, and Paint Shop, among other software” to design for the Sims after starting with a miniscule amount of technical knowledge: “she didn’t know how to turn a PC on” (Gee, 2011, 93). Izard reported that she was motivated to learn because “I was amazed with the little people on the monitor...I loved playing the game but my first love was making the homes for the little people to live in” (Gee, 2011, 94). Her interest in building Sims homes eventually led to an interest in architecture
and landscaping. Both Tabby Lou and Izaru made incredible strides in teaching themselves how to manipulate software because they were enthusiastic about changing the synthetic environment of an avatar, whether for themselves or for someone they cared about. If this kind of devotion could be applied to a traditional classroom environment, it seems likely that the shortages of STEM-educated workers would be reduced substantially.

What *The Sims* created for these women was a passion for technical work and STEM. Traditional learning environments were far less effective in creating or developing this passion for them. In Jade’s case, the traditional classroom actively discouraged her. The question is why these women developed this same passion by customizing representations.

**The Benefit of Avatars and Representation**

**The Proteus Effect**

Studies have shown that when a simulated world allows for a customized avatar, it not only engrosses the player/user, it also can transform their identity. This section will expand on the literature relating to avatars and representation in digital worlds.

Nick Yee coined the term *Proteus Effect* to describe how those participating in a synthetic environment take on the sociological characteristics of their representative avatars. In his research, Yee explains how the Proteus effect transforms online behavior thusly:

Users who are deindividuated in online environments may adhere to a new identity that is inferred from their avatars. And in the same way that subjects in black uniforms conform to a more aggressive identity, users in online environments may conform to the expectations and stereotypes of the identity of their avatars. Or more precisely, in line with self-perception theory, they conform to the behavior that they believe others would expect them to have. We term this the Proteus Effect. (Yee, 2007, 7)
Yee tested this theory by having subjects interact in an online environment while using “attractive” avatars. He found that those with the attractive avatars were shown to direct their avatars much closer in proximity to their confederates and acted in a friendlier manner than those with self-perceived unattractive representations (Yee, 2007, 16). These findings match previous sociological studies that show that women who perceive themselves as attractive tend to act in a friendlier manner than those who do not (ibid). After manipulating the height of the avatars, a similar finding was reached; as in studies where taller individuals tend to act more aggressively and confidently, taller avatars tended to respond to a negotiation task with more confidence than did those with shorter avatars (Yee, 2007, 21). The users came to identify with their avatars to such an extent that it overwhelmed their previous sociological conditioning. This is a sign of just how powerful the bond between avatars and users can become.

The Proteus Effect, as defined by Yee, reveals both the speed and the extent to which users come to identify with their avatars. In the first study, which compared attractive avatars with ugly ones, the shift in behavior came about rapidly. Participants only just had time to absorb their online physical appearance. Concomitantly, an undesirable appearance can just as quickly lead to the wholesale rejection of a synthetic world. In one 2008 topographical study, Yee argues that the infamous gender disparity in certain online games is not due to innate disinterest in mechanics for women but a perceived clash in culture, one which originates in the design of the character creator. One woman in particular cites the design of an elf character in World of Warcraft as alienating, though she otherwise enjoys gameplay:

The only really off-putting detail is that it’s ludicrous that every time my elf fights, her breasts stick out to the side repeatedly. It is a constant reminder to me
that this game is made for 13 year old boys, or men who still think like them. (World of Warcraft, female, 42) (Yee, 2008, 93)

These two studies suggest that an avatar as a form of digital self-representation can have a powerful effect on the player/user. An avatar with whom the user identifies can transform behavior, while an unacceptable avatar is alienating enough to break immersion or ruin the synthetic experience.

The Why of Whyville

Yasmin Kafei has written on the use of avatars in contemporary online learning environments, revealing that the students trend just as much toward customization and changing their form of representation as they do to gameplay. In her 2008 study on the website Whyville.com, a virtual world for science learning and exploration, Kafei investigated how avatar customization enhances the user experience (See Figure 7).

She discovered that over a third of Whyville participants are digitally located in areas devoted to customizing avatars or trading and selling avatar parts; indeed, these areas are the most trafficked part of the Whyville virtual environment (Kafai, 2008, 585). In the same study, more than fifty percent of participants admitted to spending more than half of their virtual currency on customizing their face (Kafai, 2008, 586). Qualitative data on player offline interaction showed a similar interest, as participants actively sought advice and collaborated on how to improve avatar appearance via asynchronous online media (Kafai, 2008, 587). In her analysis Kafei suggests the focus on avatar-related activities is actually beneficial to science education:
Through a quantitative lens, it is clear that these trends in Whyville use are robust. However, a definitive interpretation of the data is ambiguous. It is possible that Whyville.net was designed in such a way that users must navigate through a disproportionate number of pages to accomplish minimal avatar-related tasks. However, it is also possible that avatar activity is a highly valued aspect of social and civic life in Whyville that drives engagement independent of users’ desire to alter their avatars’ appearance. By integrating the quantitative and qualitative data, however, the evidence strongly suggests that the latter interpretation is the correct one.

Despite the flurry of activity surrounding its avatars, Whyville separates these avatar-related areas from the spaces on the website that provide science lessons and educational environments. Both Whyville and World of Warcraft connect avatar customization with monetary rewards. In a sense, therefore, completing certain lucrative tasks within the game allows the user greater freedom in changing avatar appearance. In the case of World of Warcraft, a cottage industry has formed around paying others US Dollars, Euros, and so forth, to collect the virtual coinage needed for added strength and more customization options (Dibbell, 2007: See Figure 8). This type of enthusiasm for avatars can be directly connected to the learning experience, as it contributes to creating a more active learning environment. The rest of the paper will explore this idea.
THE PAPER DOLL PROJECT

Methods and Structure

From here, this paper will move into the Paper Doll Project. The first step of the project involved coding a small learning simulation, based on visually explaining essential concepts that allow object-oriented code to function. Object-oriented programming, or OOP, is an essential aspect of modern programming languages. OOP allows for more complex pieces of code to be linked and categorized, providing a more human-readable program. Moreover, OOP allows for discrete parts of the code to perform related tasks. For example, in a traditional shooter video game, the avatar, the bullets, and the targets are all objects. With object-oriented programming, each object can execute different actions independent of the others. Casey Reas, one of the designers of the Processing language, explains the importance of object-oriented programming in this way:

Object-oriented programming is a different way of thinking about programming...In addition to providing a helpful conceptual model, object-oriented programming becomes a necessity when a program includes many elements or when it grows larger than a few pages of code. (Reas, 2007, 395).

Without object-oriented programming, it would be difficult to create programs more complex than the Logo chair example. A computer programming language, or a programmer, that does not have the capacity to work with objects will be challenged to build and maintain even a moderately complex program.

The program created for this project is a learning simulation with some interactive qualities. The avatar (coded to look like a blank slate, with only the barest of details
differentiating the male “Alan” and the female “Ada” avatars) embodies visual examples of important object-oriented programming concepts. For example, to teach the concept of instantiation, which allows multiple copies of one object to exist, the user can press a button to see many copies of the doll object appear on screen in random locations. The code acts as a learning environment, rather than the interface or the software itself. The students will customize the code to reflect their own personal interests. This allows the student to create a visual hack – they transform the appearance of the avatar as it performs these educational simulations.

_Hacking_ is here defined as taking an old piece of code and reworking it for the hacker’s own purpose. This definition comes from Mackenzie Wark’s writing, in particular his image of the hacker as a creator. Here, Wark reflects on the significance of hacking to the modern world:

> Whatever code we hack, be it programming language, poetic language, math or music, curves or colorings, we create the possibility of new things entering the world. Not always great things, or even good things, but new things. In art, in science, in philosophy and culture, in any production of knowledge where data can be gathered, where information can be extracted from it, and where in that information new possibilities for the world are produced, there are hackers hacking the new out of the old. (Wark, 2004, loc. 002)

To Wark, hacking is both a form of creative expression and a way to distill new information from old data. Too frequently, hacking is misconstrued as cracking. The _cracker_ is a hacker who uses his or her programming knowledge to steal information or commit crime. It is likely that largely because of this negative association, _hacking_ has largely been ignored as a learning tool. In this project, however, hacking will be understood to be the _de facto_ student activity.

The objective of this project is two-fold. There are two research questions that the project hopes to pursue:
1. When the learner is confronted with a self-representation coding project, what do they try to code? Is their goal a doppelgänger, some fantastic or idealized version of themselves, or something entirely different?

2. Does the visual hack of the avatar encourage the learner to take agency and customize the code itself? Do they feel more empowered and more enthusiastic about learning the advanced programming techniques necessary to build more complicated avatars?

The study is entirely qualitative, consisting of personal interviews with the subjects as well as an analysis their code. Each participant has been given a month to hack the original code. In creating this situation, my goal has been to put the ideas of John Seely Brown, Seymour Papert, and ultimately Jean Piaget into practice: To create an active learning environment for learning code and procedural thinking. Within this active learning environment, learners can absorb the lessons of object-oriented programming by focusing on a self-representative and customizable image, similar to an avatar.

The project began by coding the program that students would use to practice hacking. The coding process took approximately five months. The focus was on creating a “hackable” avatar. As discussed by Yee and Kafei, games like The Sims, Whyville and Second Life have made building an avatar an essential aspect of gameplay, which in some cases drives the initial play experience. In this case, the students will be “hacking” the avatars, instead of interacting with a custom-built interface, and thereby work directly with object-oriented code itself. After the program was completed, the code was then introduced to the graduate students of Professor Garrison LeMasters’ Expressive Computation class, CCTP764 Spring 2013, a graduate course
within the Georgetown University School of Arts and Sciences. Those who opted into the study then hacked the program, changing the way the avatar looked based on their own personal interests. There were offered no credit or compensation for participation; the students discussed in the case studies volunteered to participate with this knowledge.

Two surveys were given to the participants. The initial survey was designed to discover whether or not the students showed any initial interest in learning code, hacking, or building applications. At the end of the semester, the students filled out another survey, one which was designed to ascertain if the students felt inspired by the project and if they enjoyed learning with the Processing language, the concept of visual hacking, and/or the Paper Dolls Project itself. Students also provided feedback at that point and addressed any concerns they may have developed over the course of the project.

The qualitative data was collected through in-person interviews and emails. Each student was encouraged to check in whenever possible; most did so on a weekly basis. They were encouraged to send samples of code and ask questions during the hacking process, but they were not required to do so. Because the students worked independently and outside of a classroom environment, it is impossible to compare the length of time spent coding. Furthermore, because of how differently each student approached the project, it is impossible to say which students were more engaged or enthusiastic.

**Processing and the Coding of the Aesthetic**

The Paper Doll Project is written in Processing. Processing is a Java-based programming language designed by Casey Reas and Ben Fry. The language is specifically designed to blur the
lines between traditional computer science and artistic expression, and therefore invite visual artists to use programming as a medium for their expressive installations. The creators describe the language in this way:

Processing is a free, open source programming language and environment used by students, artists, designers, architects, researchers, and hobbyists for learning, prototyping, and production. Processing is developed by artists and designers as an alternative to proprietary software tools in the same domain. The project integrates a programming language, development environment, and teaching methodology into a unified structure for learning and exploration. (Reas, 2007, xxii)

Reas instructs novice programmers to code as often as possible for one reason: “just as it’s not possible to learn to cook without cooking, it’s not possible to learn how to program without programming” (Reas, 2007, xxii). In this way, the language was developed with the active learning environment built directly into its system. The language was also written specifically to take advantage of an online community of learners: The language is available for free online to take advantage of “continuous conversation” among a robust online community dedicated to the use and improvement of the software (Reas, 2007, xxvi). Finally, the language was also designed to “teach fundamentals of computer programming within a visual context” (Reas, 2007, 1). In this way, it lends itself to teaching code through visual metaphors, similar to Papert’s Logo language.

Processing is designed to read like a human language, and thus make it relatively easy for inexperienced or novice programmers to work with code immediately. This design provides an active learning environment quickly and reduces abstraction. Even better, the language allows programmers easily to create complex visual output. In other words, this programming language is perfect for studying how students code a visual project. Its robust web presence allows for a
wide variety of learning communities, such as the Computational Expression “classroom” on OpenProcessing (http://www.openprocessing.org/). These communities, like the Sims-based communities Gee studied, are natural environments for peer-to-peer learning. Processing, in short, seems like an ideal language to study visual hacking and modern computer science learning.

**Coding and Procedural Art: The Building of the Paper Dolls**

To quote Paul Graham, “Great software requires a fanatical devotion to beauty. If you look inside good software, you find that parts no one is ever supposed to see are beautiful too.” (Graham, 2003). Coding is both an art and a craft. It is an iterative process of planning, composing, and then fixing errors, like art. Art is iterative. Art takes the artist into unexpected places. And even though the experience of creating art is rewarding, art can sometimes frustrate and humiliate. Beautiful working code requires focus, patience, and most importantly, creativity.

Figure 9:
The code in this project presents the user with six screens. Each screen uses an avatar to help to visualize simple interactive lessons on object-oriented programming. The first screens define object-oriented programming. After that, the screens depict key computer science terms. Instantiation, as discussed previously, is represented by pressing an on-screen button to make multiple copies of one object appear; see Figure 9 for an image. After instantiation there is a screen for learning how variables work within an object. In this section, students can manipulate one avatar to change its variables while two others remain static (See Figure 10). The final section is about methods. Methods are customized pieces of code that allow an object to perform a function or an action independent of all other objects. On the methods screen, students manipulate buttons that show the avatar performing the various methods that the programmer can customize (See Figure 11). On each screen the same avatar appears: a blank slate with no defining features. For the hacker/participant, the challenge of this exercise is to manipulate the appearance of the avatar while maintaining its functionality.

Programming began in October 2012. The focus was on building an avatar whose defining features, such as the width of the waist and the size of the head, could be “pulled” on, thereby using an interactive image to illustrate the transformation of variables. This initial code was meant as a proof of concept for the avatar project overall. At this point, it was decided that volunteer participation in the project should coincide with the unit on Object-Oriented Programming in the Computational Expression class.

Code, like all crafts, is a laborious process. Like prose, code has rules that must be followed – grammar, syntax, punctuation. But unlike prose, where the human mind can work
around errors like split infinitives and comma splices, the computer will typically refuse to execute a line of code with imperfect grammar or unconventional spelling. As an example, here is a sample of code from the project:

```java
Doll(String sex, color tone, float xSpot, float ySpot, float diamX, float diamY) {
    //this is the constructor, it assures us that the value assignments we give in the
    //drawing code will associate with the proper variables
    gender = sex;
    brainX = xSpot;
    brainY = ySpot;
    diameterX = diamX;
    diameterY = diamY;
    shade = tone;
}
```

This segment of code, which occurs within a Class definition (for the Doll class), is called the constructor. It allows an object in Processing to have variables that can define its behavior. With the constructor, users can predetermine a number of features of the doll. In this case, the code allows the user to pick a gender and a name (male for Alan, female for Ada), the color of the doll, the placement of the center of the doll’s head, and the height and width of the ellipse that makes up the doll’s head. At first glance, the code this segment appears to be repetitive and unnecessary – it essentially takes a series of variables and gives them a new name. But without adhering to this specific syntax, the object would not function. Like a period at the end of a sentence, this snippet of code is essential for the computer to understand how the object looks and behaves.

Building this application was a lengthy process. One part of the initial code, the curvature of the waist, took two weeks to lay out properly – half of that time was spent getting something to appear on screen, while the other half was spent fine-tuning the curve to resemble an
hourglass figure. The time-consuming aspects of development are important to consider because, as discussed in the methodology, the students themselves will have only one-fifth of the time it actually took to create the original code.

Coding is a craft, requiring significant practice and patience. For an effective image, consider this excerpt from Stephen King’s *On Writing*, where he describes how James Joyce composed his famous prose:

“A friend came to visit James Joyce one day and found the great man sprawled across his writing desk in a posture of utter despair. James, what’s wrong?” the friend asked. ‘Is it the work?’ Joyce indicated assent without even raising his head to look at his friend. Of course it was the work; isn’t it always? How many words did you get today?” the friend pursued. Joyce (still in despair, still sprawled face down on his desk): ‘Seven.’ Seven? But James… that’s good, at least for you.’ ‘Yes,’ Joyce said, finally looking up. ‘I suppose it is… but I don’t know what order they go in!”

Just as in writing, in coding small differences in order make an extraordinary difference. The position of the FILL command, which changes the color of objects on screen, can determine if all of the objects on screen turn blue or just one. Consequently, even small mistakes can make catastrophic errors.

The completed program is relatively simplistic. It is designed this way for two reasons: to make the code easier for the novice programmer to understand, and because more code means greater chance for error. To make things as accessible as possible for the participants, students were instructed to attend exclusively to one section of the code (the segment that actually draws the doll on the screen), and to ignore the rest.
THE STUDY

Learner Profiles

Eight students agreed to participate in the thesis project. They decided to do so with the knowledge that their grades would not be affected by this choice. They were free to leave the program at any time without penalty – one student chose to do so almost immediately, and another did so later on. Only a few students had any background experience in computer languages or computational thinking. All of the subjects are in their twenties and chose to take the Computational Expression class purely out of elective interest in learning how to program. They generally showed more interest in building their own creations than manipulating the programs of others. At the beginning of this study, they had completed eight weeks of Computational Expression. By that point, they all had learned the basics of coding and knew the concepts behind object-oriented programming, but had little experience writing object-oriented code. Of the six participants, there are three men and three women. Two of the participants speak English as a second language.

It should be noted that a significant limitation of this study is that it cannot accurately determine if it adds to the enthusiasm of the participants, as the students have already in most cases demonstrated enthusiasm for the subject. Indeed, many of the participants in this study regularly spent time after class working independently or collaboratively on their coding projects. These projects, even prior to learning object-oriented programming, were sophisticated and showed evidence of creativity and independent research. At the same time, at the beginning of the study, the participants did express some confusion regarding objects and object-oriented programming. It is likely, then, that an unintended consequence of this project was to help the
participants better understand material that may not have been sufficiently addressed in the course proper.
CASE STUDIES

Initial Instructions and Caveats

The first week of the project was during Georgetown’s Spring Break. The first assignment was simple: come up with an idea of what you would like to change about the paper doll. The students were explicitly given the option of sharing their ideas or submitting them privately. The weeks following Spring Break were spent coding, with some samples sent by e-mail over the course of the four week program. Volunteers’ final projects were submitted during the first week of April. Significantly, two separate University holidays made it difficult to keep in constant touch with the students and monitor their progress.

Rose and the Self-Reflexive Avatar

The first subject, referred to as Rose², is a second-year CCT student. She is one of the few to come to Computational Expression with some previous computer science experience. Her high school taught BASIC, or Beginner's All-purpose Symbolic Instruction Code, a general-purpose language designed in the 1960s. She also knows some HTML. She reports that her primary interest is in building her own applications and expanding her coding knowledge. Other than complimenting the pedagogy of her current curriculum, she has no suggestions on how to improve her learning experience.

When asked what she would like to focus on Rose listed a number of aesthetic elements she planned to transform, mostly relating to the avatar’s clothes:

I'd like to give my doll (Ada) a hat, eyes and eyelashes, eyebrows, a nose, and a mouth. I'd also like to change her dress color (maybe add some polka dots?) and give her a belt.
Gradually, Rose worked to improve the facial features of her doll, and then began to experiment with changing the clothes and adding hair. Rose reported in interviews that the code overwhelmed her at first but the visual element helped her substantially. When asked what part of the avatar was her first priority, Rose said it was the face: "I wanted to give it a face because then it's like a person," she reported. In interviews and correspondence, Rose was quick to identify with her doll.

At the conclusion of the unit, Rose was one of the few who completed most of her desired changes. In interviews she expressed enthusiasm about her project and was eager to show off what she created. When pressed, she admitted that was disappointed that she did not reach all of her stated goals: She was unable to create a necklace or a belt, for example, due to the complexity of the body code. Nor could she add the polka dots she initially considered. She was also unhappy that a problem with her code meant that the face itself did not grow and shrink with
the rest of the head on the variable transformation screen. She reported that she based the project "somewhat" upon of herself, with brunette hair and clothing that resembled her preferred style.

Rose approached the project bit by bit each week. She described starting each coding session with the expectation of adding one small thing, and that she became increasingly excited about customizing the avatar. Over time she became inspired to try and make her project look better. She also changed the name of the doll as it appears on the screen, from "Ada" to "Eloise." Rose described herself as investing considerable time in making her doll conform to her initial proposal as closely as possible.

Rose’s avatar is a pale young woman with a red bow, brown hair, a blue dress and red shoes. Considerable detail has been added to the face, such as eyes, eyebrows, blushed cheeks and distinctly feminine lips. Her changes to the code focused on aesthetics: she did not add any new interactive elements. With regard to other parts of the doll, Rose manipulated the color and the thickness of the outline. All of these changes to the code meant that the doll’s dress, face, hands and feet could all be a different color, suggesting that Rose understood variables, even if her deployment of them was somewhat uneven: because the sizes of the individual shapes remained absolute values, the lower hair “bobs” would not transform even if manipulated in the variable section. Nevertheless, both Rose’s interviews and her code show an enthusiastic person invested in a project.

```plaintext
noStroke();
fill(75, 46, 17);
ellipse(brainX, brainY, diameterX+50, diameterY+80);
ellipse(brainX-55, brainY+40, 70, 70);
ellipse(brainX+55, brainY+40, 70, 70);

In this example, Rose chooses to use red, green, and blue (RGB) values to change the color of various shapes on the screen. When I asked her about the process which led to her colors, she confirmed that she used trial and error to discover each shade.

//neck
stroke(0);
fill(252, 243, 209); // ROSE COLOR CHANGE
```
beginShape();
  //ROSE COLOR CHANGE
  fill(36, 38, 129);
  // left curve
  Here, the original body code is left unchanged. Rose admitted she had trouble with the
code for curves and could not figure out how to change values in that shape without contorting
the body. At the same time, the hands, like the feet, show evidence of careful color manipulation.
After reading the code through, Rose reported, she understood how the curvature code worked,
but did not feel comfortable hacking it due to its apparent complexity.

  //hands
  //rose color change
  fill(252, 243, 209);
  ellipse(leftHandX, leftHandY, handSize, handSize);
  ellipse(rightHandX, rightHandY, handSize, handSize);
  //feet
  //rose color change
  fill(203, 33, 39);
  ellipse(leftFootX, leftFootY, footSize, footSize);
  ellipse(rightFootX, rightFootY, footSize, footSize);

  In this example, Rose has changed the head of the doll significantly. In the original code,
there is a single ellipse. To add extra features, Rose had to add a series of primitives on top of the
original circle. The code for the mouth is particularly complex; Rose changed the transparency of
the shapes, reducing the saturation of the color in an isolated area to better blend skin and lip
shades. She also carefully overlapped a series of circles to draw a perfectly shaped pout. This
entire section is solely Rose’s code:

  //head
  //rose color change
  fill(252, 243, 209);
Rose is an ideal case – she took ownership and created a narrative around her avatar, working to imitate her own personal style in code form. She expressed enthusiasm throughout the semester and was the most consistently in touch about the project. If the unit had been longer, she likely would have been able to conquer her issues with the body code and add the extra details she desired.

In the exit survey Rose displayed unwavering enthusiasm for her project’s coding. She rated the visual aspects of the code and the customizable elements of her avatar with the highest possible score. When asked to reflect on her project, she showed some obvious satisfaction with the final results of her work:
I might not be the coding master or be able to just whip up an app at the drop of the hat, but I feel comfortable reading and editing code and understanding the formatting/layout of how to code. Now it’s just time to practice, practice, practice!

When asked to reflect on what she liked about the project, she pointed to the creative aspects, the flexibility, and the feeling of ownership she had toward the project. She preferred starting with a piece of working code, she said, because she could then focus on personalizing it and seeing how her changes worked, rather than starting from nothing and just hoping for something to appear on the screen.

The paper doll assignment was refreshing in that I felt like I, for the most part, had control over my work as it was something I could execute with greater fluidity. I got to focus on actually making it look how I wanted it to look rather than just hoping something might work at some point. Also, I love customizable projects in general, particularly those that involve a graphic (and fashion!) component. It’s a fun way to express myself.

Rose did criticize the lack of guidance, however. She was particularly confused about the various variables and functions with which she was unfamiliar. This could have been resolved by giving the student only the parts of the program that she needed to transform directly.

I think a little table with some of the variables would be useful, particularly for those with less coding experience than myself. I was able to figure most of them out eventually, but some guidance on that would have made the process even better.

While Rose felt that her project was a success, and that she learned a great deal from the experience, she noted there are areas to improve: She appreciated the freedom of the project, but still desired guidance and help. In a more traditional learning environment, with a readily accessible teacher or professor, Rose might have been able to accomplish even more.
Eric and the Star Wars Saga Hack

Eric is one of the few students with more interest in hacking than in building his own programs. He’s a twenty year-old first-year grad student with no previous experience in computer language learning. He mentions on his initial survey that prefers a “communal” form of learning, what Douglas Thomas and John Seely Brown refer to as peer-to-peer learning.

Instead of a self-representation, Eric shows interest in portraying a character. In his proposal, Eric explained that he wanted to turn his avatar into a character from the popular movie Star Wars. In this way, his plan distances itself not just from self-reflexive qualities, such as those Rose used in her program, but from the human itself: Eric wants to build an alien.

I think right now I'm going to try and build Admiral Akbar with some trappy (sic) quality of some sort. I'm thinking that you can interact with elements that warn you of traps or general star warsy strategery (sic).

Figure 14:

Due to personal issues, Eric did not have as much time as the others did to produce his project. Still, he was interested in creating the Star Wars character to the degree he could. For
this reason he focused on changing the color of the doll to match that of the *Stars Wars*
character, playing with the original code in order to match the orange skin of the fish-like alien.

When I spoke with Eric regarding the project he emphasized how important it was to find the
right orange to match the alien, and the time he spent finding the right shade for project.

```cpp
fill(shade);
//hands
ellipse(leftHandX, leftHandY, handSize, handSize);
ellipse(rightHandX, rightHandY, handSize, handSize);
//feet
ellipse(leftFootX, leftFootY, footSize, footSize);
ellipse(rightFootX, rightFootY, footSize, footSize);
//head
ellipse(brainX, brainY, diameterX, diameterY);//eric head
```

He also changed the code in order to allow for both the orange skin tone of the character
and the white clothing, evoking the robes of the character. This action, as was the case with
Rose’s code, involved moving around segments of code based on where he wanted to put the
colors he wanted and adding new color code just for the clothes.

```cpp
beginShape();
fill(#FFFFFF);// left curve
vertex(leftCurveStartX, leftCurveStartY); //sets the left shoulder
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
curveVertex(leftCurveStartX, leftCurveStartY); //first actual point
curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5);//hip
vertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //guiding point
// right curve
vertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //right bottom point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //guiding point
curveVertex(rightCurveStartX-innerCurve,
rightCurveStartY+torsoLength*.5);//hip
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
```

54
curveVertex(rightCurveStartX, rightCurveStartY); // guiding point
vertex(rightCurveStartX, rightCurveStartY); // sets the right shoulder
// give the shoulders a small amount of curve... hopefully
curveVertex(rightCurveStartX, rightCurveStartY); // guiding point
curveVertex(rightCurveStartX, rightCurveStartY);
curveVertex(brainX, rightCurveStartY-2); // puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
curveVertex(leftCurveStartX, leftCurveStartY); // guiding point
vertex(leftCurveStartX, leftCurveStartY); // finishes the shape using the first point
endShape();
// bottom half
rectMode(CORNER);
rect(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5, shoulders-
innerCurve*2, bottomHalf);
fill(shade);

He also changed the name of the doll almost immediately to “Admiral Ackbar,” a sign of how much he valued identifying his doll as a character that inspired his imagination.

String male = "Admiral Ackbar";// eric

Eric’s code showed an understanding of variables and the syntax required to incorporate multiple colors. Still, Eric expressed disappointment with his inability to devote more time to the project. He said in his exit survey that he enjoyed the course and the chance to practice simple programming concepts. He generally would have preferred more time and perhaps a different system of organization. The problems expressed themselves early on in the study. He sought out help during the first few weeks of coding because he was not seeing his changes appear on the screen when he ran through the program. After some time he ascertained that he had been making his changes to the female code, when he hoped to change the male code. As a consequence Eric lost time and enthusiasm for the project. This frustration was later expressed in
his exit survey. However, he still felt enthusiasm for the Processing as a language and the project as a whole, giving it the highest rating in every category.

Improvements were mostly needed on my part. Though I did realize I had trouble following the flow of the program because it wasn’t organized in a way that I would have organized the code. I was cognizant that I had trouble following the logic.

Eric’s unique situation aside, he probably would have benefitted from more direct guidance and less code to filter through. Coding in the complex facial features of an alien would take significantly more time than he originally recognized. He would have needed more time or more assistance to fulfill his stated goal.

**Belle and the Easter Egg Hack**

Belle is a twenty-three year-old first year grad student. She lacks any previous experience in computer science. When asked about why she decided to sign up for Computational Expression, she said that she wants to learn coding and “have fun” at the same time. Her interest is more muted than Rose and Eric. Like them, creating new programs interests her more than hacking.

Belle’s ideas were not as ambitious as Rose’s in terms of aesthetics. In fact, other than the mouth and general “body type” she had no comments about what the avatar would look like once she finished the project. However, she showed more interest in animation, interactivity, and creating new methods, which is more ambitious than the instructions required:

I looked through your code, and I think I'll probably play around with distinctions in body types between Ada and Alan. Also, it would be kind of cool if I could add a mouth that the user can manipulate somehow. I'll make some time to write out the pseudocode this weekend.
Like Rose, Belle was consistently in-touch about her process over the course of the month. Her core interest was interactivity over aesthetics: when I noticed that she was working with the male code, she admitted the only reason she was doing so was that male was the default, and she didn't feel interested in changing it either way. She was engaged enough by adding interactive elements; adding customized visual elements was a secondary priority.

Belle created a very different hack from the others. Her remix involved changing the doll's expression using various keyboard commands. Other than that, she changed nothing about how the doll appeared. Belle built four specific states in total which can be changed with the press of a button (Figures 15, 16, 17 and 18). When I discussed her final version with her, she admitted she wanted to make more faces, and that she had not had time to write instructions directly on the screen. She characterized her project as an “Easter egg,” a popular software term for something hidden within the code that only those in the know can find. Although her changes
look small, the effort she put into creating a secret, alternate form of interactivity makes a great
difference to the code.

```java
//head
ellipse(brainX, brainY, diameterX, diameterY);
if (key == 'n') {
    ellipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
    ellipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
    line(brainX-5, brainY+20, brainX+5, brainY+20);
}
if (key == 's') {
    ellipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
    ellipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
    line(brainX-35, brainY-25, brainX-25, brainY-15);
    line(brainX+35, brainY-25, brainX+25, brainY-15);
    ellipse(brainX, brainY+10, diameterX/2, diameterY/3);
}
if (key == 'h') {
    ellipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
    ellipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
    line(brainX-35, brainY-25, brainX-25, brainY-25);
    line(brainX+35, brainY-25, brainX+25, brainY-25);
    triangle(brainX-35, brainY+15, brainX+25, brainY+15, brainX, brainY+40);
}
if (key == 'e') {
    ellipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
    ellipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
    line(brainX-35, brainY-25, brainX-25, brainY-15);
    line(brainX+35, brainY-25, brainX+25, brainY-15);
    triangle(brainX-35, brainY+15, brainX+25, brainY+15, brainX, brainY+40);
}

Belle’s code indicates a familiarity with variables and advanced interactivity functions.

While she did not change the features to the extent that Rose did or focus on colors the way that
Eric did, Belle did add an interactive component that neither attempted. She also avoided
glitches, such as Rose’s issues with the variable screen. Because everything in Belle’s code is linked proportionally to the initial ellipse, the variable screen allows the facial features to change along with the head size, meaning there are a limited number of bugs.

In her final survey, Belle wrote that she felt that her skills improved thanks to the project. She was generally pleased with her creation.

I feel like I have demonstrated competency in that I have written code that works and looks the way I wanted it to.

Belle liked that the open nature of the project let her work on what interested her most. She rated the language of Processing and the Paper Dolls program very highly, with her responses staying in the 4-5 range.

I enjoyed the Paper Dolls segment because it gave me more practice with user interactivity commands using the keyboard.

Her concerns with the project were very similar to Rose's. She was confused by some of the variables that the program utilized. She also would have preferred a more guided experience.

I think a little table with some of the variables would be useful, particularly for those with less coding experience than myself. I was able to figure most of them out eventually, but some guidance on that would have made the process even better.

Belle showed a clear understanding of the code and was able to manipulate it effectively. Because she did not have Rose's interest in fashion, she didn't focus on changing the avatar's clothes or name. However, she did develop ownership of her ideas and created a project idea that both appealed to her and was within her range of skill. Her narrow focus on creating an Easter egg involving the face allowed her to complete her project with a minimum of frustration while staying deeply engaged. Her case shows that even if the student is not drawn to the on-
representation of the self or the narrative built by the code, they can still create a deep encounter by utilizing their creativity.

**Philip and the Radical Reworking Hack**

Philip is a twenty-two year-old first-year. He has previously worked with HTML in an informal, self-taught environment. Other than that, he has no experience in programming. He expressed interest in user interfaces in his initial survey, specifically in what goes on behind the glossy screen. As with Rose, he is more interested in building his own products rather than taking apart what someone else has built. Also like Rose, he has no ideas on how to make computer science learning more appealing. English is his second language.

When he wrote in his proposal, he focused on his key interest, human-computer interaction:

Cool doll! I would like to communicate with the doll, like in a role playing-game. Say that it asks me to take a stance on an issue, I respond, and the doll then responds to my response, etc. Basically, to carry a conversation with the doll!

When interviewed about his planned project, Philip admitted his main desire was to manipulate the code outside of the object-oriented programming framework, even if it would require more work from him personally. His main desire was to realize the prompting element – he was far more interested in interactivity over creating facial features or otherwise transforming the form of the doll. His ideas, in the end, proved too ambitious; when he researched procedures for user text input via Processing, he discovered it was a more difficult idea to realize than he had originally anticipated. Though he found examples, he was uncomfortable with the coding required, and thus was unable to do the project he really wanted to do.
Because of the time spent researching what he was ultimately unable to accomplish, Philip could not hand-in a complete code project at the end of the unit. Instead of building a working version, he created a mock-up for of his desired design. Notably, Philip did not use object-oriented programming to build his mockup: While his doll is visually identical to the original doll, he actually rewrote the program completely. He also imported images to symbolize his workaround for the prompting issue: the code has two buttons, a 1 and a 2. According to Philip, the program would, when functional, allow the doll to respond to Yes or No questions.

Philip does use some advanced functions, such as importing and displaying images, but does not utilize other types of variables. Arguably, he showed the least improvement in his understanding and implementation of code. However, this is due (at least in part) to the time spent in research; it is likely that with a less ambitious proposal, he might have spent more time in the coding environment.
PImage img1, img2;
void setup() {
    size(600,600);
    background(0);
    img1 = loadImage("1_tracy.png");
    img2 = loadImage("2_tracy.png");
}
void draw() {
    fill(100);
    ellipse(350,500,50,50);
    ellipse(250,500,50,50);
    ellipse(400,280,50,50);
    ellipse(200,280,50,50);
    rect(280,300,40,100);
    rect(285,100,30,100);
    quad(220, 200, 380, 200, 320, 340, 280, 340);
    ellipse(300,130,100,100);
    image(img1, 100, 50);
    image(img2, 400, 50);}

While Philip successfully avoided implementing OOP, he showed impressive ambition, searching out a way to code a project beyond the basic aesthetic changes he was encouraged to apply. Like Belle, he found an alternate form of engagement when he was less interested in a purely visual hack. The difference is that because his ideas were so transformational and ambitious, he failed to complete his project. While Belle was satisfied and proud, Philip expressed constant frustration.

In sum, Philip created what I call a radical reworking. Besides moving the doll into a completely different environment, he reworked the program's original purpose. He changed the colors and some of the relative ratios between upper and lower body. He also removed the name. When asked to reflect on his progress, Philip admitted that he was not able to write advanced Processing code. He did feel he was becoming more code-literate working with the project and,
had he more time, he would have had a better chance accomplishing his initial objective. His response, however, reveals his ongoing sense of frustration:

I have demonstrated that I have limited knowledge in working with someone else's code. Of course, I wouldn't be able to produce the advanced code you had us work with--however, by looking at it, I can imitate the appearance of it. Have I had more time and less work on my own plate, I might have been able to thoroughly go through and familiarize with your code.

Like Rose, his main complaint with the unit was the lack of guidance. Having too much freedom put more pressure on him than a more "narrow" assignment might have allowed.

When learning something new, I am someone who needs a lot of guidance--especially when learning a new language. This class had some, but for me to fully be able to grasp it, I would have needed more time on the basics, more narrow assignments, etc. to guide me to where the professor would want me to be. The free structure where the students have a lot of freedom to create what they'd like is great for someone with more basic knowledge of code than what I had.

Ultimately, Philip had more muted enthusiasm than Rose did, and suggested ultimately that the project would have been better done with a more advanced level of computer science proficiency. The fact he was unable to build his desired program within the allotted time frame frustrated and angered him rather than deepened his previous enthusiasm for programming languages. The openness of the assignment was too much, he felt, for a beginner.

I feel like working with someone else's code might have been more suitable in a second semester course.

Perhaps with a greater amount of time, or even with a different language more open to prompting through text, the project would have been more successful. His desire to create such a complex project shows an initial level of excitement that could have been deepened with more mentoring or more time.
Ariel and the Remix Hack

Ariel is another CCT second-year grad student with limited computer science training. She decided to take Computational Expression purely to “learn something new.” English is not her first language, so over the course of the unit she had some difficulty expressing herself when discussing the project. However, despite her lack of previous experience she proved to be an adept coder with an eye for design. Her plan was a unique type of hack – a type of remix project.

Ariel chose to combine an existing class project with the assignment. Earlier in the semester, she wrote a program which made a series of identical cats walk across the screen. This is the code she refers to in her project proposal.

I want the change the face of the doll to make it look like a cat (just like what I did for one of our assignment (sic.)). The cat can move its limbs and dance like a doll. The color of the doll will also be changed.

Ariel is the only student interested in using code she had already built. In a sense, her project is a remix as well as a hack. It is similar to a phenomenon in music, where one artist “samples” the work of another and then deploys it in a broader creative context. Although intellectual property laws generally frown upon “remix culture,” it is generally accepted within artistic contexts (see Vrana, 2011). In terms of coding, a remix is simply another kind of a hack. By remixing, Ariel can incorporate a larger quantity of her own code into her doll. But because the code transformation will be distinct, she also has a greater chance of affecting the other segments she is not responsible for.
Of the original code only the hands and feet remain, and even these have been somewhat manipulated. Every other attribute— the body, the head, the nose, and the other facial features – has been completely replaced with code from the cat project. Because Ariel was working with and improving existing code, she felt comfortable using advanced techniques, such as the Matrix. The graphics Matrix is a set of functions that allow the software to save the state of the screen, allow the user to manipulate the screen as she sees fit, and then return the screen back to its previous state. This allows for a more targeted use of transformational functions like ROTATE and SCALE. Ariel also used an alternative method of creating a curved shape to make the cat’s torso.

```java
pushMatrix();
scale(1.8);
translate(-270,-250);
stroke(0, 0, 0);
fill(0, 0, 0);
//face
ellipse(brainX+100, brainY+150, 100, 62);
fill(0, 0, 0);
```
Because Ariel incorporated so much unique code, she had difficulty adjusting her doll to fit the initial program. However, her adjustments show a deep interest in and attention to detail. She moved the hands and feet up to match the new cat body without affecting the existing "wave" code. She was very bothered that the variable screen of her program was now defunct. Because the size of the cat’s facial features are not a proportion of the cat head but static numbers, her cat’s face cannot be stretched out or otherwise changed. We worked together to discover a solution but were unable to do so before the conclusion of the unit. Another glitch lowers the cat's left paw below its body, but only during the methods screen. The cat can still perform the three dance functions as requested. The fact that glitches and bugs so bothered Ariel that she consistently sought out assistance both online and in person suggests that she felt a deep connection with the project.

Ariel also had ambitious plans beyond the conclusion of the unit. When I spoke to Ariel at the conclusion of the unit she told me that she wanted to add methods for the cat to swipe at the viewer. She also wanted to change the paws to have more cat-like aesthetic attributes. Finally, she was still frustrated with the bugs that affected interactive elements of the original Paper Doll Project. The large number of things she still wanted to change and work on indicate an enduring passion for her cat doll.
Like Rose, Ariel wrote on her exit survey about the sense of freedom that the Paper Dolls project gave to her. She found the freedom exciting and enjoyable.

I can combine what I have learned from class to the project. I also enjoy the freedom I have to design my dolls.

Overall she rated the language, the visual nature of the language, and the hacking project highly. Her suggestions came mainly in the form of ideal changes to the project code instead of pedagogy.

1. Allow people to change the size and color of the doll

2. Add more actions to the doll.

Ariel’s interest in extending the initial code may indicate some discomfort with making these changes herself. She did manage to change the size and color of the doll itself. She repeatedly said that she wanted to program the cat to perform more cat-like actions, which is why she probably wishes the doll came with more methods to manipulate.

**Flynn and the Experienced Hacker**

Flynn was a special case for several reasons. First, he was the only participant not from the CCT program. He was also one of the most educated in programming and computer science. He had previous experience with Python, a similar object-oriented programming language. In this way he was more adept than others in building objects, and already knew the information taught through the initial simulation.
The most experienced programmer had both one of the shortest descriptions and one of
the most ambitious. Instead of specifying particular elements, he had one overall vision for his
avatar, one which at first glance would require a great deal of extra coding. He also indicates
interest in building a pet for his avatar, something that would have to be done completely from
scratch.

I would like my avatar to be a pirate with an interactive parrot that flies around
him and tells bawdy jokes.

Figure 21:

Flynn's code, like many of the others, primarily focused on changing facial features. He
gave his doll a hat and added details using imported images. He also gave the doll an eye patch
to go with the pirate aesthetic and changed the thickness of the outlines.

fill(0000);
triangle(brainX -75, brainY, brainX +75, brainY, brainX, brainY -diameterY);
PImage img;
img = loadImage("jolly.png");
imageMode(CENTER);
image(img, brainX, brainY - 25,50,50 );
ellipse(brainX -25, brainY+10, 20, 20);
He was also one of the few to manipulate the body section of the code, changing some points in order to give the body a more extreme waist. The trapezoid was transformed into two triangles. When asked why he said he preferred this body shape. He also increased the size of the bottom half of the body to account for the change in the chest. These changes are significant: no other student felt they could attempt to manipulate this part of the code. Flynn’s background made him more confident in his ability to change a difficult part of the code to match his sensibilities.

```
beginShape();
// left curve
vertex(leftCurveStartX, leftCurveStartY); //sets the left shoulder
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
curveVertex(leftCurveStartX, leftCurveStartY); //first actual point
curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5);//hip
vertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //guiding point
// right curve
vertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //right bottom point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //guiding point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //hip
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
vertex(rightCurveStartX, rightCurveStartY); //sets the right shoulder
//give the shoulders a small amount of curve...hopefully
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //setting point
curveVertex(brainX, rightCurveStartY-2); //puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
```
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
vertex(leftCurveStartX, leftCurveStartY); //finishes the shape using the first point
endShape();
//bottom half
rectMode(CORNER);
rect(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5, shoulders-innerCurve*2, bottomHalf);

Flynn, the most experienced programmer, showed both a strong grasp of variables and more advanced techniques such as importing images. He was also the only one to read through and understand the curveVertex and vertex code, changing it around to match his preferences. He admitted disappointment that he was not able to incorporate the parrot he initially desired, but reported satisfaction overall with the project he managed to construct.

Flynn’s previous experience also helped him to articulate where he felt the project and the language could use some improvement. He was more enamored with the visual aspects of Processing than the language itself or the project. He also suggested some revisions with the code when asked to reflect on the past month.

The code could have been laid out in a way that made it more readable/moddable. I also found it difficult to just interact with the dolls without struggling to find a way out of the tutorial.

His extensive background made the hacking project, aimed for beginners, above his skill set. His observations about the program, however, show that in working with the project Flynn was able to practice his understand of coding languages and improve upon his already diverse skill set.

General Patterns
The sample size of the study is too small to make conclusions about general effectiveness. Still, it is worth noting some patterns which emerged over the course of the unit.
Overall the responses to the project were positive. However, several students realized their ambitions were unsustainable for the amount of time spent with the code.

Several people expressed feeling overwhelmed with the amount of code given to them, even though they knew they were only responsible for working with one small section. This indicates a feeling of responsibility with the entire project and a sense of ownership; they did not feel that they could only look at one small part, but wanted to understand how the entire program worked. In future iterations, keeping this in mind, it might be wiser to separate the code into parts and introduce units one at a time, in order to avoid overwhelming the students.

The subjects started with minor adjustments as they learned to work with the code. In general, the first changes were aesthetic and focused on creating facial features. In the words of Belle, she felt she needed to “put a face on” her avatar. Rose approached the problem the same way, and her project reflects this instinct. In the case of Eric, he was one of the few to change the name of his character immediately. This is likely due to the narrative context in which he had placed the project. The frustrations that the students expressed over time signals that a hacking project might better separate the code into parts, giving students access only to what is relevant to their experiences. Those that used the male version of the doll, for example, might have avoided a significant amount of frustration had they not had to deal with the female code irrelevant to his project.

Those who were more interested in interactivity generally felt more frustration with the project than those who focused on aesthetics. Rose, who never planned on adding any extra interactive elements, was also one of the most punctual coders and the one most satisfied with
her final product. Her micro-focus helped her avoid frustration and aided in her learning experience. Instead of absolute openness in creating the assignment, it might have functioned better by taking aesthetic ownership before letting them go beyond the scope of their initial assignment. Philip, for example, was so focused on a specific type of computer-human interaction he never put much time into changing how the doll looked. Belle is the exception – she experimented with interactivity and was satisfied. However, she, like Rose, had a laser focus. Belle only changed facial features. In keeping to one interactive aspect, one that she was both interested in and knew she could conquer, Belle’s project was well-developed and she expressed more satisfaction with the project.

Code remixing proved to be an effective learning strategy. Ariel’s dynamic project highlights the worth of this avenue. Her initial idea to combine her already existing cat project with her paper doll meant that her doll showed the most customization. The cat face, drawn exclusively with Processing primitives, dramatically changed the appearance of the doll. However, her use of unique code caused glitches in other areas, such as the variables screen, to an extent beyond what the others interacted with. Her removal of the original torso code meant that the pulling methods no longer functioned. She wanted to fix the glitch, but since her cat body did not have the necessary variable, she felt at a loss. A future example of the project will need to account for this problem and provide more mentorship to account for this.

When meeting with the students, the most enlightening discussions came in groups. Rose, who felt comfortable with the code, advised the more frustrated Eric and Philip to consider downgrading their high ambitions and focus on the look of their avatars. She understood that her
focus on aesthetic aspects helping to sustain her passion for her project. Other students chose to speak one-on-one or through email alone; while we were able to work on problems with the primary investigator, they could not benefit from the experience of their fellow students, which had helped others to alleviate some concerns. This suggests that a face-to-face, peer-to-peer element is an effective complement to hacking.
CONCLUSION

Due to the sample size, it is impossible to draw any absolute conclusions as to whether or not hacking works better than more traditional forms of computer science learning. However, the Paper Doll Project does show that hacking can be an effective learning tool, and that digital representations can engender passion in learners.

The students involved with the project all expressed enthusiasm, particularly with the visual elements of the project. Each took a completely different approach to the singular question – how will you change this existing piece of code? One person, Rose, chose to make a doll that reflected herself. Belle chose to create an interactive Easter egg, giving her doll a number of faces to try on. Ariel remixed her code, incorporating elements from a previous project to create a doll that was completely and utterly hers. Philip completely reworked the code to pursue his own vision. Eric took inspiration from a popular character, taking ownership through a fantastical narrative. Flynn developed a character through the use of outside imagery to create a pirate character.

All of the participants expressed a desire to continue working, pointing to parts that they wanted to complete and objectives they had to abandon due to time constraints. This indicates an enthusiasm and ownership of their projects beyond simple homework. Every student learned more about the coding language through a through reading of the code involved, even if they did not understand every single point. Freedom came up multiple times: the Piaget-style “learning without a curriculum” approach seemed to instill a sense of freedom in the participants. Some, such as Rose and Ariel, relished this freedom; others, such as Philip, felt a narrower project with
tighter constraints would have benefitted him more. Flynn, who had a stronger background in computer science, did not get as much from the project as his less experienced peers; this suggests that the less one has to learn from the existing code, the less compelling the project might be for the learner.

Hacking can work within a curriculum, with the proper implementation. The code should be a level above the competency of the students but not too high; it must be well-commented; and it should point towards an open goal to invoke the sense of freedom the subjects relished. The decision to use a cartoonish human, a blank slate, encouraged imagination from all of the students, and they exercised their creativity as well as their coding abilities. In this way, the project successfully met some of the goals that Papert set forth when he described an effective computer science learning environment. The students were enthusiastic and intrigued by the concepts illustrated in the Paper Doll Project. The only caveat is that to avoid frustration more direct intervention by a master or teacher should be considered.

For teachers considering hacking as a pedagogical tool, a few caveats must be considered. Creating a “hackable” program requires a significant time commitment. A project needs to be developed and tested before it can be introduced to students. If students do not comprehend the code, even some small aspects, they can become deeply frustrated. But a visual hack, particularly one with a customizable element like an avatar, can work.

It will be interesting to see, as computation continues to be part of our daily lives, if future schools take learning programming languages more seriously. Hacking might be a tool for these classrooms. Since this project used adult graduate students, it would also be interesting to
see how younger children might handle working not with a shiny interface, but directly with code.

Hacking has taken on unfortunate connotations in recent years, as the popular image has shifted from creative coders into the more sinister, black-hatted figure. But hacking is a form of creative expression, one that could be better utilized in the classroom. It is time to embrace, rather than discourage, the natural desire to rework and remix, to work collaboratively, and to find agency in coding creatively. It might be worth it just to see what a new generation of creative coders can build.
END NOTES

1. Gamification is defined as “the application of online game design techniques in non-game setting” (Gaggioli, 2012)
2. Please note that all of the names of the participants have been changed to protect their anonymity.
APPENDIX A: EXAMPLE SYLLABI

Syllabus A

General Course Description and Objectives:
AP Computer Science: goes beyond merely learning how to use applications like word processing, spread sheets, and internet browsers. It uses the Java language and focuses on the basic principles needed to design and build applications. It's a college level course. At the course's end, students will be required to take the AP Computer Science A test. If they pass they may receive college credit for one semester of computer science.

Course Outline

I. Unit Title:

Software & Hardware Basics

Begin and End Dates:
08 -19, 09 - 18

Chapters:
1, 2

Specific Outcomes (Objectives/Standards):
AP Computer Science Standard: II Program Implementati on, III Program Analysis, VI Computing in Context,
I

Object-Oriented Program Design

Unit Assessment:

Test
II. Unit Title:
Introduction to Classes & Objects

Begin and End Dates:
09 – 21, 10 - 02

Chapters:
3

Specific Outcomes (Objectives/Standards)
Object-Oriented Program Design, II Program Implementation

Unit Assessment:
Test

III. Unit Title:
Algorithms

Begin and End Dates:
10 – 05, 10 - 16

Chapters:
4

Specific Outcomes (Objectives/Standards)
Program Analysis

Unit Assessment:
Test

IV. Unit Title:
Java Syntax and Style,
Data Types, Variables, and Arithmetic

Begin and End Dates:
10 – 19, 10 - 30

Chapters:
5 & 6

Specific Outcomes (Objectives/Standards)
Program Implementation, III Program Analysis

Unit Assessment:
Test

V. Unit Title:

Boolean Expressions and Conditional Control

Begin and End Dates:
11 – 02, 11 - 13

Chapters:
7

Specific Outcomes (Objectives/Standards)
Program Implementation, III Program Analysis

Unit Assessment:
Test

VI. Unit Title:

Iterative Statements (Loops)

Begin and End Dates:
11 – 16, 12 - 04

Chapters:
8

Specific Outcomes (Objectives/Standards)

Program Implementation, III Program Analysis

Unit Assessment:

Test

VII. Unit Title:
Implementing Classes and Using Objects

Begin and End Dates:
01 – 04, 12 - 18

Chapters:
9

Specific Outcomes (Objectives/Standards)

II Program Implementation, III Program Analysis

Unit Assessment:

Test

VIII. Unit Title:
Strings

Begin and End Dates:
01 – 20, 02 - 02

Chapters:
Specific Outcomes (Objectives/Standards)

II. Program Implementation, III. Program Analysis

Unit Assessment:
Test

IX. Unit Title:
Class Hierarchies and Interfaces

Begin and End Dates:
02 – 03, 02 - 18 - 10

Chapters:
11

Specific Outcomes (Objectives/Standards)

I. Object-Oriented Program Design

Unit Assessment:
Test

X. Unit Title:
Arrays and ArrayLists

Begin and End Dates:
02 - 11 – 10, 02 - 18 - 10

Chapters:
13

Specific Outcomes (Objectives/Standards)
II Program Implementation, III Program Analysis

Unit Assessment:

Test

XI. Unit Title:

Searching Sorting and Other Array Elements

Begin and End Dates:

02 – 19, 03 - 19

Chapters:

12

Specific Outcomes (Objectives/Standards)

Unit Assessment:

Test

XII. Unit Title:

Begin and End Dates:

Chapters:

Specific Outcomes (Objectives/Standards)

Unit Assessment:

Test

XIII. Unit Title:

Personal Project

Begin and End Dates:

12 – 07, 05 - 03
Chapters:

10 & 11

Specific Outcomes (Objectives/Standards)

Design and create a significant software project

Unit Assessment:

Test

Text:

Java Methods A & AB


Materials Needed:

- A USB thumb drive or other storage media for maintaining your electronic portfolio of physics assignments. We will attempt to be as close to a paperless classroom as possible.
- A set of dry erase markers. You will frequently be working problems in class on a white board.
- A package of 3x5 cards: Starting immediately, each student will, over the course of the year create a set of flash cards to use as a study aid.
- A graphing calculator

Grading Policy and Assessments:

A = 93-100

B = 85-92
C = 77-84
D = 70-76
F = 0-69

Quarter:

Minor assessments will consist of:

- Tests and major projects
- Major assessments will consist of:
  - Homework, participation, labs, minor projects

Attendance Policy:

School Policy: a student may not miss more than ten days from a year-long course. Those ten days include parent’s notes, suspensions, unexcused absences, administrative, or late arrival notes. After ten absences, a doctor’s note or administrative excuse must be provided or the student will not receive credit for the course.

What to do if you miss a class:

Excused Absence:

Quizzes cannot be made up but will not count against a student with an excused absence. If you have an excused absence, you will be able to make up all other work. Provision for make-up work is the student’s responsibility and must be done outside of class within five (5) consecutive school days after the student returns to school.

Unexcused Absence:

Make up work and tests for unexcused absences will not be accepted.

1. Academic and Behavioral Expectation
2. If Mr. X, a guest speaker, or a substitute is addressing the class or a test is in progress, students should be silent. Otherwise, students may discuss class related information in low level voices. The noise level should never rise to the point that it is hard to hear.

3. Remain seated except with teacher permission.

4. Come to class prepared and use your class time for learning the subject.

5. All equipment in the classroom is off limits except with teacher permission.

6. Commit yourself to passing the AP Exam.

7. Be respectful to others at all times, especially to guests and visitors.

Syllabus B

CS110

Introduction to Programming

Syllabus

Textbook:

Using C++: An Introduction to Programming


Auxiliary Material: Lab exercises, practice quizzes, and programming assignments will be made available online (on the Blackboard system)

Course Description

This course is designed to introduce students to the craft of computer programming. The goal is for you to get a sense of how to make computers (as stupid mechanisms) do things that we would like them to do, and at the same time to see how computers (as intelligent machines) can do
things that even surprises the very people who program them. Except for basic computer skills such as keyboarding, the course does not presuppose any prior knowledge of computers or of programming on your part. By the end of the semester, you should have a basic understanding of programming concepts and constructs such as numbers, strings, assignments, sequential-versus-selective execution, nesting, loops, functions, arrays, reference parameters, file streams, etc.

The course includes a lab session that takes place once a week. Lab projects involve programming exercises that could be typically completed during the lab session.

Grading

The final grade will be based on

- Programming assignments (400)
- Quizzes (100)
- Two midterms (300)
- Final exam (200)

4H Principles: Hard work, honesty, helpfulness, and humility throughout the semester and in relation to others are the major principles of this class, and will be rewarded in the course.

Participation:

Students are expected to take active part in class and lab discussions by paying close attention, raising questions, making suggestions, posing challenges, casting doubt, sharing experience, and so on — basically, any means that demonstrates their interest and enthusiasm but does not violate the Standards of Academic Honesty (pages 13–20 of the catalogue). Taking advantage of instructor office hours is strongly recommended.

Programming assignments
(50 points each) will be given on (most) Mondays, and are due one week later. As a bonus, the lowest-grade assignment will not be counted in the final grade. All programming assignments are done individually.

Late hand-in is not accepted, except under documented emergency situations.

The two midterms will be taken on Feb. 11, March 10. The final is scheduled for April xx.

Schedule

<table>
<thead>
<tr>
<th>Period</th>
<th>Topic</th>
<th>Reading</th>
<th>Milestones</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS110</td>
<td>Introduction to Programming</td>
<td>Chapters 1 and 2</td>
<td>Visual Studio</td>
</tr>
<tr>
<td></td>
<td>Fall 2003</td>
<td>Visual Studio</td>
<td>Week 2: Jan. 19–23</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Quiz 1/Program 1</td>
<td>Numeric Data Types</td>
</tr>
</tbody>
</table>
Week 3: Jan.26–30
Selective Execution
Chapter 4 &12
Quiz 2/Program 2

Week 4: Feb. 2– Feb. 6
Functions
Chapter 5
Quiz 3/ Program 3

Week 5: Feb. 9–13
Strings
Chapter 6
Test 1 (W: 02/11)

Week 6: Feb. 16-20
Loops
Chapter 7
Quiz 4/ Program 4

Week 7: Feb. 23–27
Nested Loops
Chapter 8
Quiz 5/Program 5

March 01–05
Spring Break
Week 8: March 08–12
Reference Parameters, etc.
Chapter 9 & 10
Test 2 (W: 03/10)
Week 9: March 15–19
File Streams
Chapter 11
Quiz 6/Program 6
Week 10: March 22-26
Arrays & Vectors
Chapter 13
Quiz 7/Program 7
Week 11: March 29–April 2
Structs
Chapter 17
Quiz 8/Program 8
Week 12: April 05–09
Classes
Chapter 18
Test 3 (M: 04/05)
Week 13: April 12–16
CodeAcademy is a dynamic website. While screenshots have been taken of individual assignments, for a complete look at the syllabus go to
http://www.codecademy.com/tracks/javascript. Included is a sampling of their Javascript syllabus.

JavaScript

1 Introduction to JavaScript

An introduction to JavaScript, a beginner-friendly programming language.

   Getting Started with Programming Leng Lee 28 Exercises

   100%

   Project: Choose Your Own Adventure! Leng Lee 7 Exercises

2 Functions
Functions store blocks of code that you can later call at any time to avoid repetition in your program.

Introduction to Functions in JS Leng Lee 13 Exercises

Project: Build "Rock, Paper, Scissors" Leng Lee 1 / 9 Exercises

11%

Project: Review of Functions in JavaScript Albert Wenger 8 Exercises

3 'For' Loops in JavaScript

'For' loops are a series of instructions that repeat until a condition is met.

Introduction to 'For' Loops in JS Leng Lee 14 Exercises

Project: Search Text for Your Name Eric Weinstein 7 Exercises

4 'While' Loops in JavaScript

While loops are useful when you want to loop through a program without knowing ahead of time how many loops you'll need to make.

Introduction to 'While' Loops in JS Eric Weinstein 11 Exercises

Project: Dragon Slayer! Eric Weinstein 6 Exercises

5 Control Flow

Conditionals let your program make choices to execute different parts of your code.

More on Control Flow in JS Eric Weinstein 14 Exercises

Project: Choose Your Own Adventure 2! Eric Weinstein 6 Exercises

6 Data Structures

Arrays and objects are more complex data structures that will allow us to write more powerful programs.
Arrays and Objects in JS Eric Weinstein 17 Exercises

Project: Contact List Eric Weinstein 8 Exercises

7 Objects I

Objects are a data type that lets us store real world information within a single structure.

They're the building blocks of object-oriented programming (OOP).

Introduction to Objects I Spencer de Mars 33 Exercises

Project: Building an Address Book Leng Lee 6 Exercises

8 Objects II

Continue to learn more about objects, and how to use them in a variety of situations!

Introduction to Objects II David Hu 30 Exercises

Project: Building a Cash Register Shaun Forsyth
APPENDIX B: CODE

The Paper Doll Project Master Code

/* OpenProcessing Tweak of *%@*http://www.openprocessing.org/sketch/79178*%@* */
/* !do not delete the line above, required for linking your tweak if you re-upload */
//Paper Dolls - Alternate Version - now with OOP power

//I consulted the following while writing this code:
//Processing: A Programming Handbook for Visual Designers and Artists, by Casey Reas and
Ben Fry (2007)

//any numbers needed
float placeX, placeY; //used in randomizing the placement of dolls during the instances screen
int instanceNum = 199; // max number of instances during the instances screen - the higher the
number, the more memory required, but the more fun you can have mashing the make one button
//declare our objects
Doll a, b, c, methodMan; // declare our object
Doll[] random = new Doll[instanceNum]; //this array will create randomly placed instances to
fill the screen during the instantiation

//declare the types of screens we will have
boolean intro, define, defineTwo, instances, arguments, methods; // each of these arguments (and
 corresponding screens) represents a lesson
Screen text;
boolean[] instanceExists = new boolean[instanceNum];

//declares any buttons that will appear on the screen
Button letsMoveOn, makeInstance;

//these arrays will let you make various dance moves
int danceNum = 3;
String[] danceText = new String[danceNum];
Button[] danceMoves = new Button[danceNum];

//these arrays will let you review previous material
int reviewScreens = 4;
String[] reviewType = new String[reviewScreens];
Button[] reviewButton = new Button[reviewScreens];

//define any fonts
PFont bigFont, smallFont;
//These two strings determine the shape of your representation.
//Choose as you will - feel free to change the name
String male = "Alan";
String female = "Ada";
String moveOn = "Continue";
String instance = "Make One";

//declare any other variables
float mashCounter; //this counter will help us go through the different flags

//global variables for the variable screen

void setup() {
    //size of screen
    size(800, 600);
    // smooth out lines
    smooth();
    //provides the color of the background
    background(240);
    //sets up any fonts
    bigFont = createFont("Arial", 36, true);
    smallFont = createFont("Arial", 24, true);
    //assign variables for our object - change the variables to change the basic look of the ball &
    //where it is positioned
    //use male or female to determine gender and name - anything else will confuse the code.
    a = new Doll(male, #B92525, width/2, height/2, 100, 100);
    b = new Doll(male, #236FE5, width/2, height/2, 100, 100);
    c = new Doll(male, #236FE5, width/2, height/2, 100, 100);
    methodMan = new Doll(male, #B92525, width/2, height/2, 100, 100);
    text = new Screen(male, bigFont, smallFont);
    letsMoveOn = new Button (width-100, height-100, 100, 50, #7B64FF, moveOn);
    makeInstance = new Button (width*.5, height*.6, 120, 120, #AE2EFF, instance);
    //everything related the methods/dance screen
    danceText[0] = "To the Left";
    danceText[1] = "To the Right";
    danceMoves[0] = new Button (width*.2, height*.53, 180, 70, #AE2EFF, danceText[0]);
    danceMoves[1] = new Button (width*.2, height*.73, 180, 70, #AE2EFF, danceText[1]);
    //initializing button color
    danceMoves[0].setupColor();
    danceMoves[1].setupColor();
    danceMoves[2].setupColor();
}
makeInstance.setupColor();
//setting up review buttons & initializing color
reviewType[0] = "Introduction";
reviewType[1] = "Instances";
reviewType[2] = "Variables";
reviewType[3] = "Methods";
reviewButton[0] = new Button (width*.35, height*.5, 180, 120, #AE2EFF, reviewType[0]);
reviewButton[1] = new Button (width*.35, height*.75, 180, 120, #AE2EFF, reviewType[1]);
reviewButton[2] = new Button (width*.65, height*.5, 180, 120, #AE2EFF, reviewType[2]);
reviewButton[3] = new Button (width*.65, height*.75, 180, 120, #AE2EFF, reviewType[3]);
reviewButton[0].setupColor();
reviewButton[1].setupColor();
reviewButton[2].setupColor();
reviewButton[3].setupColor();
//setting up variables so that they are given proper values (meaning they only are valued as this
once and can be changed later using methods)
a.setupDoll();
b.setupDoll();
c.setupDoll();
a.waveSetup();
a.variableSetup();
}

void draw() {
  background(240);
  //letsMoveOn.buttonDraw();
  //draw the doll and any text screens relating to the doll
  if (mashCounter==0) { //this draws the intro screen
text.writeIntro();
pushMatrix();
scale(.5);
translate(width/2, -height/4);
a.wave();
popMatrix();
}
if (mashCounter==1) { //this draws the first screen defining OOP
text.defineOOP1();
pushMatrix();
translate(-width/4, -height/8);
a.wave();
popMatrix();
}
if (mashCounter==2) {
pushMatrix();
translate(-width/4, -height/8);
a.wave();
popMatrix();
text.defineOOP2();
}
if (mashCounter ==3) { //in this screen a button lets you create an unlimited number of dolls by
pressing the create button, graphically explaining instances

text.instances();
makeInstance.buttonDraw();
for (int i = 1; i<makeInstance.mashCounter(); i = i+1 ) {
    random[i].make();
}
makeInstance.buttonDraw();
}
if (mashCounter ==4) {
    makeInstance.reset();
pushMatrix();
scale(.75, .75);
translate(width*.55, height*.15);
b.make();
popMatrix();
pushMatrix();
scale(.75, .75);
translate(width*.25, height*.15);
c.make();
popMatrix();
a.drawVariables();
text.variables();
}
if (mashCounter==5) {

text.methods();
println(danceMoves[0].mashCounter() + " and " + danceMoves[1].mashCounter() + " and "+danceMoves[2].mashCounter());
if (danceMoves[0].mashCounter()%2 ==0) {
    if (danceMoves[1].mashCounter()%2 ==0) {
        if (danceMoves[2].mashCounter()%2 ==0) {
            danceMoves[0].buttonDraw();
danceMoves[1].buttonDraw();
danceMoves[2].buttonDraw();
pushMatrix();
scale(.6, .6);
translate(width*.52, height*.49);
//methodMan.clearChanges();
methodMan.make();
popMatrix();
danceMoves[0].bumpCounter();
danceMoves[1].bumpCounter();
danceMoves[2].bumpCounter();
methodMan.dancingIsForbidden();
}
}

if (danceMoves[0].mashCounter() > 1 && danceMoves[0].mashCounter() % 2 == 1) {
pushMatrix();
scale(.6, .6);
translate(width*.52, height*.49);
methodMan.toTheLeft();
popMatrix();
danceMoves[0].turnOn();
danceMoves[1].buttonDraw();
danceMoves[2].buttonDraw();
danceMoves[1].putBabyInACorner();
danceMoves[2].putBabyInACorner();
}
else if (danceMoves[1].mashCounter() > 1 && danceMoves[1].mashCounter() % 2 == 1) {
pushMatrix();
scale(.6, .6);
translate(width*.52, height*.49);
methodMan.toTheRight();
popMatrix();
danceMoves[1].turnOn();
danceMoves[0].buttonDraw();
danceMoves[2].buttonDraw();
danceMoves[0].putBabyInACorner();
danceMoves[2].putBabyInACorner();
}
else if (danceMoves[2].mashCounter() > 1 && danceMoves[2].mashCounter() % 2 == 1) {
pushMatrix();
scale(.6, .6);
translate(width*.52, height*.49);
methodMan.nowKick();
popMatrix();
danceMoves[2].turnOn();
danceMoves[0].buttonDraw();
danceMoves[1].buttonDraw();
danceMoves[1].putBabyInACorner();
danceMoves[0].putBabyInACorner();
}
}
if (mashCounter == 6) {
text.finishHim();
reviewButton[0].buttonDraw();
reviewButton[1].buttonDraw();
reviewButton[2].buttonDraw();
reviewButton[3].buttonDraw();
if (reviewButton[0].mashCounter()>=1) {
    mashCounter = 0;
    reviewButton[0].reset();
    reviewButton[1].reset();
    reviewButton[2].reset();
    reviewButton[3].reset();
}
if (reviewButton[1].mashCounter()>=1) {
    mashCounter=3;
    reviewButton[0].reset();
    reviewButton[1].reset();
    reviewButton[2].reset();
    reviewButton[3].reset();
}
if (reviewButton[2].mashCounter()>=1) {
    mashCounter=4;
    reviewButton[0].reset();
    reviewButton[1].reset();
    reviewButton[2].reset();
    reviewButton[3].reset();
}
if (reviewButton[3].mashCounter()>=1) {
    mashCounter = 5;
    reviewButton[0].reset();
    reviewButton[1].reset();
    reviewButton[2].reset();
    reviewButton[3].reset();
}
if (mashCounter>6) {
    mashCounter = 6;
    reviewButton[0].reset();
    reviewButton[1].reset();
    reviewButton[2].reset();
reviewButton[3].reset();
}
}

// press M and you continue in the program
void keyReleased() {
    if ((key == 'm' || key == 'M')) {
        mashCounter++;
    }
}

void mouseReleased() {
    makeInstance.buttonMash();
    if (mashCounter==3) {
        instanceExists[makeInstance.mashCounter()] = true;
        randomizePlacement();
        random[makeInstance.mashCounter()] = new Doll(male, #B92525, placeX, placeY, 100, 100);
        random[makeInstance.mashCounter()].setupDoll();
    }
    if (mashCounter==5) {
        // a.dancingIsForbidden();
        // danceMoves[0].putBabyInACorner();
        // danceMoves[1].putBabyInACorner();
        // danceMoves[2].putBabyInACorner();
    }
    if (mashCounter==6) {

        if (mouseX<width*.5) {

            if (mouseY<height*.65) {

                reviewButton[0].buttonMash();
            } else {
                reviewButton[1].buttonMash();
            }
        }
        else {
            if (mouseY<height*.65) {
                reviewButton[2].buttonMash();
            } else {
                reviewButton[3].buttonMash();
            }
        }
    }
}
void randomizePlacement() {
    placeX = random(0, width);
    placeY = random(height/2, height);
}

void mouseDragged() {
    if (mashCounter ==4) {
        if (mousePressed) {
            //waist joint
            //if (mouseX >=a.waistX() -a.pressurePoint() && mouseX<=a.waistX() +a.pressurePoint()) {
            // if (mouseY >=a.waistY() -a.pressurePoint() && mouseY<=a.waistY() +a.pressurePoint()) {
            //
            // a.dragWaist();
            // }
            // }
            //head - x values - manipulate the width of the head
            //if (mouseX >=headWidthPointX-pressurePointRad && mouseX <=headWidthPointX+pressurePointRad) {
            //    //headSizeX = brainX+headSizeX*.5-mouseX;
            //    //WidthPointX = mouseX;
            //    //}
            //}
            //head - y values - manipulate the width of the head
            if (mouseX >=a.headLengthX()-a.pressurePoint() && mouseX <=a.headLengthX()+a.pressurePoint()) {
                // if (mouseY >=a.headLengthY()-a.pressurePoint() && mouseY <=a.headLengthY()+a.pressurePoint()) {
                a.dragChin();
                // }
            }
            else if (mouseX >=a.headWidthX()-a.pressurePoint() && mouseX <=a.headWidthX()+a.pressurePoint()) {
                if (mouseY >=a.headWidthY()-a.pressurePoint() && mouseY <=a.headWidthY()+a.pressurePoint()) {
                    a.dragEar();
                }
            }
        }
    }
}
if (mouseY>=a.waistY()-a.pressurePoint() && mouseY<=a.waistY()+a.pressurePoint()) {
    a.dragWaist();
}

// }
// }
void mousePressed() {
    if (mashCounter == 5) {
        if (mouseY<height*.53+70) {
            danceMoves[0].buttonMash();
        }
        if (mouseY>height*.73-70 && mouseY<height*.73+70) {
            danceMoves[1].buttonMash();
        }
        if (mouseY>height*.93-70) {
            danceMoves[2].buttonMash();
        }
    }
}

class Button {
    float xPos, yPos, howWide, howTall;
    color shading, shade;
    String text;
    int counter = 0;
    Button(float x, float y, float xLong, float yLong, color pretty, String words) {
        xPos = x;
        yPos = y;
        howWide = xLong;
        howTall = yLong;
        shade = pretty;
        text = words;
    }

    void setupColor() {
        shading = shade;
    }

    void buttonDraw() {
        fill(shading);
        stroke(0);
    }
}
rectMode(CENTER);
rect(xPos, yPos, howWide, howTall, 7);
fill(0);
textAlign(CENTER, CENTER);
text(text, xPos, yPos);
fill(shading);
}
void buttonMash() {
  if (mouseX>xPos-howWide || mouseX<xPos+howWide) {
    if (mouseY>yPos-howTall || mouseY<yPos+howWide) {
      counter = counter+1;
    }
  }
}
int mashCounter() {
  int mashCounter = counter;
  return(mashCounter);
}
void turnOn() {
  shading = shade;

  if (counter%2 == 1) {
    shading = #CE1D1D; //red
  }
  buttonDraw();
  shading = shade;
}
void putBabyInACorner() { //no one puts baby in a corner except me - a method to stop all the
dancing by reducing all mash Counters to zero
  counter = 2;
}
void bumpCounter() { // increases the counter value so that dancing can start as soon as the first
button is pressed
  counter = 2;
}
void reset() {
  counter = 0;
}
}

//this tab will have everything relating to the doll as an object. Here you'll find the class
declaration, the constructor, all of the custom functions, everything you need to change the ball
//to change the appearance of the ball, scroll to the "make" function
//to give the doll new animations, add a new custom function
//to make the figure female, look for Ada
//to make a male figure, look for Alan

class Doll { // this is the class name
    float brainX, brainY, diameterX, diameterY; // these are all of the variables that we can change when we declare the object
    color shade;
    String gender;

    //sizing the neck
    float neckWidth; //thickness of the neck
    float neckLength; //length of the neck

    //giving the neck location variables based on the head
    float neckStartsX;
    float neckStartsY;

    //things I'd ideally like people to be able to change
    float innerCurve; //this determines how thin the waist is - the bigger the value, the more extreme the waist size
    float waistSize;

    //CURVES, how I loathe them
    float shoulders;// the width of the shoulders - bigger the value, the thicker the body
    float torsoLength; //the length of the body
    float leftCurveStartX; //leftmost point X
    float leftCurveStartY; //leftmost top point Y - the higher the value, the further the body will be from the head
    float leftCurveStopX; //leftmost point X
    float leftCurveStopY; //leftmost bottom point Y
    float rightCurveStartX; //rightmost point X
    float rightCurveStartY; //rightmost top point Y
    float rightCurveStopX; //rightmost point X
    float rightCurveStopY; //leftmost bottom point Y

    //Mii-Style hand balls
    float leftHandX;
    float rightHandX;
    float leftHandY;
    float rightHandY;
    float handSize;
//Mii-Style foot balls
float leftFootX;
float rightFootX;
float leftFootY;
float rightFootY;
float footSize;

//waists - they do not lie. Unlike hips.
float waistX = rightCurveStartX-innerCurve;
float waistY = rightCurveStartY+torsoLength*.5;
//bottom
float bottomHalf;

//dance variables
//for arms
float danceSpeed = 1;
float danceDirection = 1;
//for head and body
int bopSpeed=5;
int bopDirection=1;
//step counter
int danceStepsLeft = 0;
int danceStepsRight = 0;
//for kicking
int kickSpeed = 5;
int kickDirection = 1;
boolean kickLeft = true;
boolean kickRight = false;

//waving variables
float waveDirection = 1;
float waveSpeed = 1;

//variable stuff
//pressure points
float pressurePointRad;
//head - x value
float headWidthPointX;
float headWidthPointY;

//head - y value
float headLengthPointX;
float headLengthPointY;

//variables indicating change in values, so that variables can be reset later
float changeX, changeY;

Doll(String sex, color tone, float xSpot, float ySpot, float diamX, float diamY) {
    //this is the constructor, it assures us that the value assignments we give in the drawing code
    will associate with the proper variables
    gender = sex;
    brainX = xSpot;
    brainY = ySpot;
    diameterX = diamX;
    diameterY = diamY;
    shade = tone;
}

void setupDoll() {
    //these are variables that need declaration/definition but don't need to be individually declared
    when declaring the object
    //the value of these variables are determined based on what you declared
    //feel free to experiment with changing ratios in the setupDoll function

    //sizing the neck
    neckWidth = diameterX*.4; //thickness of the neck
    neckLength = diameterY; //length of the neck

    //giving the neck location variables based on the head
    neckStartsX = brainX;
    neckStartsY = brainY+(neckLength*.75);

    //things I'd ideally like people to be able to change
    innerCurve= diameterX*.5; //this determines how thin the waist is - the bigger the value, the
    more extreme the waist size

    //CURVES, how I loathe them
    shoulders = diameterX*1.5;// the width of the shoulders - bigger the value, the thicker the
    body
    torsoLength = diameterY*2; //the length of the body
    leftCurveStartX = brainX- shoulders*.5; //leftmost point X
    leftCurveStartY = brainY+(neckLength*.75); //leftmost top point Y - the higher the value, the
    further the body will be from the head
    leftCurveStopX = leftCurveStartX; //leftmost point X
leftCurveStopY = leftCurveStartY+torsoLength; //leftmost bottom point Y
rightCurveStartX = brainX+shoulders*.5; //rightmost point X
rightCurveStartY = leftCurveStartY; //rightmost top point Y
rightCurveStopX = rightCurveStartX; //rightmost point X
rightCurveStopY = rightCurveStartY+torsoLength; //leftmost bottom point Y

//Mii-Style hand balls
leftHandX = brainX - (shoulders*.75);
rightHandX = brainX + (shoulders*.75);
leftHandY = brainY + torsoLength*.75;
rightHandY = brainY + torsoLength*.75;
handSize = diameterX*.5;

//Mii-Style foot balls
footSize = diameterX*.6;
leftFootX = brainX - (shoulders*.35);
rightFootX = brainX + (shoulders*.35);
leftFootY = brainY + torsoLength*1.6;
rightFootY = brainY + torsoLength*1.6;

//waists - they do not lie. Unlike hips.
waistX = rightCurveStartX-innerCurve;
waistY = rightCurveStartY+torsoLength*.5;

//bottom half
bottomHalf = shoulders*.5;

//this code determines the color
fill(shade);
//this determines the color of the outline - I'm leaving it black & unchangeable, but you don't have to
stroke(0);

//pressure points
pressurePointRad=20;
//head - x value
headWidthPointX = brainX+diameterX*.5;
headWidthPointY = brainY;

//head - y value
headLengthPointX = brainX;
headLengthPointY = brainY+diameterY*.5;
void make() {

    //sets how Processing will read the ellipse and rectangle functions. Switch around and your
doll will turn Picasso
    rectMode(CENTER);
    ellipseMode(CENTER);
    fill(shade);
    stroke(0);

    //THIS IS WHERE THE DRAWING BEGINS
    //I CANNOT STRESS THIS ENOUGH

    if (gender == female) {
        //this is the female code
        //CHANGE FEMALE CHARACTERISTICS HERE

        //this draws the doll
        //neck
        rect(neckStartsX, neckStartsY, neckWidth, neckLength);

        //body

        //curves, which I hate, die curves die
        beginShape();
        // left curve
        vertex(leftCurveStartX, leftCurveStartY); //sets the left shoulder
        curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
        curveVertex(leftCurveStartX, leftCurveStartY); //first actual point
        curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //waist
        curveVertex(leftCurveStartX, leftCurveStartY); //moves back to the left
        curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
        vertex(leftCurveStartX, leftCurveStartY); //sets the bottom left point

        // right curve

        //sets value of changed variables to zero
    changeX = 0;
    changeY = 0;
    }

}
vertex(rightCurveStopX, rightCurveStopY); //right bottom point
curveVertex(rightCurveStopX, rightCurveStopY); //guiding point
curveVertex(rightCurveStopX, rightCurveStopY); //bottom right point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5);//waist
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
vertex(rightCurveStartX, rightCurveStartY); //sets the right shoulder

//give the shoulders a small amount of curve...hopefully
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
curveVertex(rightCurveStartX, rightCurveStartY);
curveVertex(brainX, rightCurveStartY-2); //puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
vertex(leftCurveStartX, leftCurveStartY); //finishes the shape using the first point
endShape();

//hands
ellipse(leftHandX, leftHandY, handSize, handSize);
ellipse(rightHandX, rightHandY, handSize, handSize);

//feet
ellipse(leftFootX, leftFootY, footSize, footSize);
ellipse(rightFootX, rightFootY, footSize, footSize);

//head
ellipse(brainX, brainY, diameterX, diameterY);
}
else {
//this is the male code

//CHANGE MALE CHARACTERISTICS HERE

//this draws the doll
//neck
rect(neckStartsX, neckStartsY, neckWidth, neckLength);

//body - the female shape focuses on making the waist, meanwhile this guy focuses on hips

//curves, which I hate, die curves die
beginShape();
// left curve
vertex(leftCurveStartX, leftCurveStartY); //sets the left shoulder
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
curveVertex(leftCurveStartX, leftCurveStartY); //first actual point
curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //hip
vertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //guiding point

// right curve
vertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //right bottom point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //guiding point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //hip
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
vertex(rightCurveStartX, rightCurveStartY); //sets the right shoulder

give the shoulders a small amount of curve...hopefully
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
curveVertex(rightCurveStartX, rightCurveStartY);
curveVertex(brainX, rightCurveStartY-2); //puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
vertex(leftCurveStartX, leftCurveStartY); //finishes the shape using the first point

endShape();

//bottom half
rectMode(CORNER);
rect(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5, shoulders-innerCurve*2, bottomHalf);

//hands
ellipse(leftHandX, leftHandY, handSize, handSize);
ellipse(rightHandX, rightHandY, handSize, handSize);

//feet
ellipse(leftFootX, leftFootY, footSize, footSize);
ellipse(rightFootX, rightFootY, footSize, footSize);
//head
ellipse(brainX, brainY, diameterX, diameterY);
void shakeHand() { //we want people to be able to grab the hand and "shake it" by holding
    make();
    if (mouseX> brainX & mouseX<brainX+200) {
      rightHandX = mouseX;
      rightHandY = mouseY;
    } else if (mouseX< brainX & mouseX>brainX-200) {
      leftHandX = mouseX;
      leftHandY = mouseY;
    }
}

void waveSetup() {
  leftHandX = brainX-100;
  leftHandY = brainY;
}

void wave() {
  make();
  leftHandX = leftHandX+waveSpeed*waveDirection;
  leftHandY = leftHandY+ waveSpeed*.5*waveDirection;
  if (leftHandX < brainX-150 || leftHandX >brainX-diameterX) {
    waveDirection=waveDirection*-1;
  }
}

void dance() {
  make();
  //putting hands in the air, like we just don't care
  leftHandX = leftHandX + danceSpeed*danceDirection;
  rightHandX = rightHandX + danceSpeed*danceDirection;
  if (leftHandX < brainX-150 || leftHandX >brainX-diameterX) {
    danceDirection=danceDirection*-1;
  }
  //whip his/her hair back and forth, whip his/her hair back and forth
  brainY = brainY +bopSpeed*bopDirection;
  if (brainY > leftCurveStartY-diameterY*.65 || brainY < neckStartsY-diameterY*.75) {
    bopDirection = bopDirection*-1;
  }
}
void variableSetup() {
    //head - x value
    headWidthPointX = brainX+diameterX*.5;
    headWidthPointY = brainY;

    //head - y value
    headLengthPointX = brainX;
    headLengthPointY = brainY+diameterY*.5;

    //waist
    waistX = rightCurveStartX-innerCurve;
    waistY = rightCurveStartY+torsoLength*.5;
}

void drawVariables() { //this method draws a doll with pressure points, which you can use to change variables
    make();
    //draw pressure points
    fill(#3A00FF);
    ellipse(waistX, waistY, pressurePointRad, pressurePointRad); //waist joint
    ellipse(headWidthPointX, headWidthPointY, pressurePointRad, pressurePointRad); //x value of head
    ellipse(headLengthPointX, headLengthPointY, pressurePointRad, pressurePointRad); //y value of head
}

void dragVariables() {
    //animation allows people to "pull" on pressure points, changing value of certain variables
    if (mouseX >=waistX-pressurePointRad && mouseX<=waistX+pressurePointRad) {
        if (mouseY >=waistY-pressurePointRad && mouseY<=waistY+pressurePointRad) {
            innerCurve = rightCurveStopX-mouseX;
            waistX = mouseX;
        }
    }
}

//head - x values - manipulate the width of the head
if (mouseX>=headWidthPointX-pressurePointRad && mouseX<=headWidthPointX+pressurePointRad) {
    if (mouseY>=headWidthPointY-pressurePointRad && mouseY<=headWidthPointY+pressurePointRad) {
        diameterX = brainX+diameterX*.5-mouseX;
        headWidthPointX = mouseX;
    }
}

113
// head - y values - manipulate the width of the head
if (mouseX >= headLengthPointX - pressurePointRad && mouseX <= headLengthPointX + pressurePointRad) {
  if (mouseY >= headLengthPointY - pressurePointRad && mouseY <= headLengthPointY + pressurePointRad) {
    diameterY = brainY + diameterY * 0.5 - mouseY;
    headLengthPointY = mouseY;
  }
}
// these are methods for the methods screen. All three will be connected to buttons to execute them at will
void toTheLeft() {
pushMatrix();

danceStepsLeft = danceStepsLeft + bopSpeed * bopDirection;

leftFootX = leftFootX - danceSpeed * danceDirection;
rightFootX = rightFootX + danceSpeed * danceDirection;
if (leftFootX < leftCurveStopX || leftFootX > brainX) {
  danceDirection = danceDirection * -1;
}
if (brainX < width * 0.35 || brainX > width * 0.9) {
  bopDirection = bopDirection * -1;
}
translate(0 - danceStepsLeft, 0);
make();
popMatrix();
}
void toTheRight() {

danceStepsRight = danceStepsRight + bopSpeed * bopDirection;
pushMatrix();

leftFootX = leftFootX + danceSpeed * danceDirection;
rightFootX = rightFootX - danceSpeed * danceDirection;
if (rightFootX > rightCurveStopX || rightFootX < brainX) {
  danceDirection = danceDirection * -1;
}
if (brainX < width * 0.35 || brainX > width * 0.9) {
bopDirection = bopDirection*-1;

translate(0+danceStepsRight, 0);
make();
popMatrix();

void nowKick() {
    make();
    if (kickLeft == true) {
        leftFootX = leftFootX - kickSpeed*kickDirection;
        leftFootY = leftFootY - kickSpeed*kickDirection;
        if (leftFootX < leftHandX || leftFootY>rightFootY) {
            kickDirection = kickDirection*-1;
        }
    } else if (kickRight == true) {
        rightFootX = rightFootX + kickSpeed*kickDirection;
        rightFootY = rightFootY - kickSpeed*kickDirection;
        if (rightFootX > rightHandX || rightFootY>leftFootY) {
            kickDirection = kickDirection*-1;
        }
    }
}

void dancingIsForbidden() {
    setupDoll();
    danceStepsLeft=0;
    danceStepsRight=0;
    //kickDirection = 1;
    if (kickLeft == true) {
        kickLeft = false;
        kickRight = true;
    } else if (kickRight = true) {
        kickLeft = true;
        kickRight = false;
    }
}

//we'll have local variables return so that we can do the mousePressed thing
float headLengthY() {
    headLengthPointY = brainY+diameterY*.5;
    return(headLengthPointY);
}

115
float headLengthX() {
    headLengthPointX = brainX;
    return(headLengthPointX);
}
float headWidthY() {
    headWidthPointY = brainY;
    return (headWidthPointY);
}
float headWidthX() {
    headWidthPointX = brainX+diameterX*.5;
    return (headWidthPointX);
}
float waistY() {
    waistY = rightCurveStartY+torsoLength*.5;
    return (waistY);
}
float waistX() {
    waistX = rightCurveStartX-innerCurve;
    return(waistX);
}
float pressurePoint() {
    return (pressurePointRad);
}
void dragChin() {
    changeY = mouseY;
    diameterY = brainY+diameterY*.5-changeY;
    headLengthPointY = mouseY;
}
void dragEar() {
    changeX = mouseX;
    diameterX = brainX+diameterX*.5-changeX;
    headWidthPointX = mouseX;
}
void dragWaist() {
    changeX = mouseX;
    innerCurve = rightCurveStopX-changeX;
    waistX = mouseX;
}
void clearChanges() {
    if(changeX > 0 || changeX < 0) {
        rotate(radians(180));
    }
    setupDoll();
}
class Screen {
    boolean type;
    PFont littleF, bigF;
    String gender;

    //universal text
    String pressMToContinue = "Press M to continue.";

    //any text on the intro screen
    String introOne = "Object-Oriented Processing";
    String introTwo = "By Tracy Carlin";
    String introThree = "Georgetown University";
    String introFour = "Spring 2013";

    //any text on the definition screen
    String defineOne = "Hi guys! Meet "+female+".";
    String defineTwo = female+" is made using";
    String defineThree = "Object-Oriented";
    String defineFour = "Programming,";
    String defineFive = "or OOP.";
    String defineSix = "OOP represents";
    String defineSeven = "concepts as 'objects,'";
    String defineEight = "which can have their";
    String defineNine = "own data and procedures.";
    String defineTen = "Let's explore";
    String defineEleven = "how they work.";

    String defineOneMale = "Hi guys! Meet "+male+".";
    String defineTwoMale = male+" is made using";

    //any text on the variance screen
    String varianceOne = "Objects work like complex custom functions.";
    String varianceTwo = "You can pass arguments";
    String varianceThree = "and create multiple variants.";
    String varianceFourFemale = "Make many "+female+"s using the button.";
    String varianceFourMale = "Make many "+male+"s using this button.";
//any text for the variable screen
String variableOne = "Each object behaves on its own.";
String variableTwoFemale = "For example, watch these "+female+"s.";
String variableTwoMale = "For example, watch these "+male+"s.";
String variableThree = "Change the variables on one.";
String variableFour = "It won't affect the other.";
String variableFive = ";
String variableSix = "";

//any text for the methods screen
String methodOne = "Objects can also execute methods.";
String methodTwo = "Each object executes customized functions";
String methodThree = "without affecting others.";
String methodFour = "Experiment with methods using these buttons.";
String methodFive = "The doll will only execute what is 'on.'";
String methodSix = "";
String methodSeven = "";

//end screen
String finishOne = "That's the basics!";
String finishTwo = "What kind of objects will you make?";
String finishThree = "Hit the buttons to review anything.";
//String finishFour
//String finishFive
//String finishSix

Screen(String sex, PFont fontUno, PFont fontDos) {
    //constructor
gender = sex;
    bigF = fontUno;
    littleF = fontDos;
    textAlign(CENTER);
}

void writeIntro() {
    fill(0);
    textAlign(CENTER);
    textFont(bigFont);
    text(introOne, width/2, height/2);
    text(introTwo, width/2, height/2+50);
    text(introThree, width/2, height/2+100);
    text(introFour, width/2, height/2+150);
void defineOOP1() {
  fill(0);
  if (gender == male) {
    textAlign(CENTER);
    text(defineOneMale, width/2, height/8);
    textAlign(RIGHT);
    text(defineTwoMale, width-50, height/3);
    text(defineThree, width-50,height/3+50);
    text(defineFour, width-50, height/3+100);
    text(defineFive, width-50, height/3+150);
    textAlign(CENTER);
    text(pressMToContinue, width/2, height-10);
  }
  if (gender == female) {
    fill(0);
    textAlign(CENTER);
    text(defineOne, width/2, height/8);
    textAlign(RIGHT);
    text(defineTwo, width-50, height/3);
    text(defineThree, width-50,height/3+50);
    text(defineFour, width-50, height/3+100);
    text(defineFive, width-50, height/3+150);
    textAlign(CENTER);
    text(pressMToContinue, width/2, height-10);
  }
}

void defineOOP2() {
  fill(0);
  textAlign(CENTER);
  text(defineSix, width*.7, height/3);
  text(defineSeven, width*.7,height/3+50);
  text(defineEight, width*.7, height/3+100);
  text(defineNine, width*.7, height/3+150);
text(defineTen, width*.7, height/3+200);
text(defineEleven, width*.7, height/3+250);
textFont(smallFont);
textAlign(CENTER);
text(pressMToContinue, width/2, height-10);
}

void instances() {
    fill(0);
    textFont(bigFont);
    textAlign(CENTER);
    text(varianceOne, width*.5, height/7);
    text(varianceTwo, width*.5, height/7+50);
    text(varianceThree, width*.5, height/7+100);
    if (gender == female) {
        text(varianceFourFemale, width*.5, height/7+150);
    }
    if (gender == male) {
        text(varianceFourMale, width*.5, height/7+150);
    }
    //text(varianceFive, width*.5, height/7+200);
    textFont(smallFont);
    textAlign(CENTER);
    text(pressMToContinue, width/2, height-10);
}

void variables() {
    fill(0);
    textFont(bigFont);
    textAlign(CENTER);
    text(variableOne, width*.5, height/7);
    if (gender == female) {
        text(variableTwoFemale, width*.5, height/7+50);
    }
    if (gender == male) {
        text(variableTwoMale, width*.5, height/7+50);
    }
    text(variableThree, width*.5, height/7+100);
    text(variableFour, width*.5, height/7+150);
    //text(variableOne, width*.5, height/7+200);
    //text(variableOne, width*.5, height/7);
    //text(variableOne, width*.5, height/7);
    textFont(smallFont);
    textAlign(CENTER);
    text(pressMToContinue, width/2, height-10);
}

120
void methods() {
    fill(0);
    textAlign(CENTER);
    textFont(bigFont);
    text(methodOne, width/2, height/8);
    text(methodTwo, width/2, height/8+50);
    text(methodThree, width/2, height/8+100);
    text(methodFour, width/2, height/8+150);
    text(methodFive, width/2, height/8+200);
    textFont(smallFont);
    text(pressMToContinue, width/2, height-10);
}

void finishHim() {
    fill(0);
    textAlign(CENTER);
    textFont(bigFont);
    text(finishOne, width*.5, height/6);
    text(finishTwo, width*.5, height/6+50);
    text(finishThree, width*.5, height/6+100);
}

Student Code: Hacks of the Paper Dolls

Note: Because the students were only responsible for changing the make() function on the doll object, I will only share that section.

Ariel

//CHANGE MALE CHARACTERISTICS HERE
fill(0,0,0);
//hands
    ellipse(leftHandX+40, leftHandY+120, handSize-10, handSize-10);
    ellipse(rightHandX-50, rightHandY, handSize-20, handSize-20);

//feet
    ellipse(leftFootX, leftFootY-100, footSize-20, footSize-20);
    ellipse(rightFootX-15, rightFootY-126, footSize-20, footSize-20);

    pushMatrix();
    scale(1.8);
curveVertex(brainX+110, brainY+210);
curveVertex(brainX+107, brainY+220);
curveVertex(brainX+105, brainY+230);
curveVertex(brainX+101, brainY+191);
curveVertex(brainX+90, brainY+181);
curveVertex(brainX+92, brainY+181);
curveVertex(brainX+60, brainY+240);
curveVertex(brainX+68, brainY+170);
endShape();
popMatrix();
}
}

Belle

//CHANGE MALE CHARACTERISTICS HERE

//this draws the doll
//neck
rect(neckStartsX, neckStartsY, neckWidth, neckLength);

//body - the female shape focuses on making the waist, meanwhile this guy focuses on hips

//curves, which I hate, die curves die
beginShape();
// left curve
vertex(leftCurveStartX, leftCurveStartY); //sets the left shoulder
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
curveVertex(leftCurveStartX, leftCurveStartY); //first actual point
curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //hip
vertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //guiding point

// right curve
vertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //right bottom point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //guiding point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //hip
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
vertex(rightCurveStartX, rightCurveStartY); //sets the right shoulder
// give the shoulders a small amount of curve...hopefully
curveVertex(rightCurveStartX, rightCurveStartY); // guiding point
curveVertex(rightCurveStartX, rightCurveStartY);
curveVertex(brainX, rightCurveStartY-2); // puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
curveVertex(leftCurveStartX, leftCurveStartY); // guiding point
vertex(leftCurveStartX, leftCurveStartY); // finishes the shape using the first point
endShape();

// bottom half
rectMode(CORNER);
rect(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5, shoulders-innerCurve*2, bottomHalf);

// hands
ellipse(leftHandX, leftHandY, handSize, handSize);
ellipse(rightHandX, rightHandY, handSize, handSize);

// feet
ellipse(leftFootX, leftFootY, footSize, footSize);
ellipse(rightFootX, rightFootY, footSize, footSize);

// head
ellipse(brainX, brainY, diameterX, diameterY);
if (key == 'n') {
elipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
elipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
line(brainX-5, brainY+20, brainX+5, brainY+20);
}
if (key == 's') {
elipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
elipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
line(brainX-35, brainY-25, brainX-25, brainY-15);
line(brainX+35, brainY-25, brainX+25, brainY-15);
elipse(brainX, brainY+10, diameterX/2, diameterY/3);
}
if (key == 'h') {
elipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
elipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
line(brainX-35, brainY-25, brainX-25, brainY-15);
line(brainX+35, brainY-25, brainX+25, brainY-25);
elipse(brainX, brainY+10, diameterX/2, diameterY/3);
}
if (key == 's') {
elipse(brainX-35, brainY+15, brainX+25, brainY+15, brainX, brainY+40);
if (key == 'e') {
    ellipse(brainX-30, brainY-5, diameterX/5, diameterY/5);
    ellipse(brainX+30, brainY-5, diameterX/5, diameterY/5);
    line(brainX-35, brainY-25, brainX-25, brainY-15);
    line(brainX+35, brainY-25, brainX+25, brainY-15);
    triangle(brainX-35, brainY+15, brainX+25, brainY+15, brainX, brainY+40);
}

fill(shade);

Rose

//CHANGE FEMALE CHARACTERISTICS HERE
//hair : rose
    noStroke();
    fill(75, 46, 17);
    ellipse(brainX, brainY, diameterX+50, diameterY+80);
    ellipse(brainX-55, brainY+40, 70, 70);
    ellipse(brainX+55, brainY+40, 70, 70);

    //this draws the doll
    //neck
    stroke(0);
    fill(252, 243, 209); // ROSE COLOR CHANGE
    rect(neckStartsX, neckStartsY, neckWidth, neckLength);

    //body

    //curves, which I hate, die curves die

    beginShape();
    //ROSE COLOR CHANGE
    fill(36, 38, 129);
    // left curve
    vertex(leftCurveStartX, leftCurveStartY); //sets the left shoulder
    curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
    curveVertex(leftCurveStartX, leftCurveStartY); //first actual point
    curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //waist

    }
curveVertex(leftCurveStopX, leftCurveStopY); //moves back to the left
curveVertex(leftCurveStopX, leftCurveStopY); //guiding point
vertex(leftCurveStopX, leftCurveStopY); //sets the bottom left point

// right curve
vertex(rightCurveStopX, rightCurveStopY); //right bottom point
curveVertex(rightCurveStopX, rightCurveStopY); //guiding point
curveVertex(rightCurveStopX, rightCurveStopY); //bottom right point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //waist
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
vertex(rightCurveStartX, rightCurveStartY); //sets the right shoulder

//give the shoulders a small amount of curve...hopefully
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
curveVertex(rightCurveStartX, rightCurveStartY);
curveVertex(brainX, rightCurveStartY-2); //puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
vertex(leftCurveStartX, leftCurveStartY); //finishes the shape using the first point
endShape();

//hands
//rose color change
fill(252, 243, 209);
ellipse(leftHandX, leftHandY, handSize, handSize);
ellipse(rightHandX, rightHandY, handSize, handSize);

//feet
//rose color change
fill(203, 33, 39);
ellipse(leftFootX, leftFootY, footSize, footSize);
ellipse(rightFootX, rightFootY, footSize, footSize);

//head
//rose color change
fill(252, 243, 209);
ellipse(brainX, brainY, diameterX, diameterY);
fill(0);
//eyes and eyebrows: ROSE
line(brainX-10, brainY-20, brainX-4, brainY-20);
line(brainX+8, brainY-20, brainX+3, brainY-20);
ellipse(brainX-6, brainY-10, 7, 7);
ellipse(brainX+6, brainY-10, 7, 7);
// mouth: ROSE
noStroke();
fill(227, 41, 75);
ellipse(brainX, brainY+15, 20, 10);
ellipse(brainX-3, brainY+13, 6, 9);
ellipse(brainX+3, brainY+13, 6, 9);
// blush: rose
fill(255, 180, 199, 78);
ellipse(brainX-25, brainY, 20, 20);
ellipse(brainX+25, brainY, 20, 20);
stroke(0);
// hair bow: rose
fill(203, 33, 39);
ellipse(brainX-30, brainY-45, 30, 25);
ellipse(brainX-3, brainY-45, 30, 25);
ellipse(brainX-15, brainY-46, 15, 15);

Flynn

// CHANGE MALE CHARACTERISTICS HERE

// this draws the doll
// neck
rect(neckStartsX, neckStartsY, neckWidth, neckLength);

// body - the female shape focuses on making the waist, meanwhile this guy focuses on hips
// curves, which I hate, die curves die
beginShape();
// left curve
vertex(leftCurveStartX, leftCurveStartY); // sets the left shoulder
curveVertex(leftCurveStartX, leftCurveStartY); // guiding point
curveVertex(leftCurveStartX, leftCurveStartY); // first actual point
curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); // hip
vertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); // guiding point

// right curve
vertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //right bottom point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //guiding point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5);//hip
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
vertex(rightCurveStartX, rightCurveStartY); //sets the right shoulder
//give the shoulders a small amount of curve...hopefully
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
curveVertex(rightCurveStartX, rightCurveStartY);
curveVertex(brainX, rightCurveStartY-2); //puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
vertex(leftCurveStartX, leftCurveStartY); //finishes the shape using the first point
endShape();

//bottom half
rectMode(CORNER);
rect(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5, shoulders-innerCurve*2, bottomHalf);

//feet
ellipse(leftFootX, leftFootY, footSize, footSize);
ellipse(rightFootX, rightFootY, footSize, footSize);
//head
ellipse(brainX, brainY, diameterX, diameterY);
fill(0000);
triangle(brainX -75, brainY, brainX +75, brainY, brainX, brainY -diameterY);
PImage img;
img = loadImage("jolly.png");
imageMode(CENTER);
image(img, brainX, brainY - 25,50,50 );

ellipse(brainX -25, brainY+10, 20, 20);
stroke(4);
line(brainX -25, brainY +10, brainX - 40, brainY + 25);
line(brainX -25, brainY +10, brainX , brainY );
fill(shade);
//hands

128
ellipse(leftHandX, leftHandY, handSize, handSize);
ellipse(rightHandX, rightHandY, handSize, handSize);
}
}

Eric

//CHANGE MALE CHARACTERISTICS HERE

//this draws the doll
//neck
rect(neckStartsX, neckStartsY, neckWidth, neckLength);

//body - the female shape focuses on making the waist, meanwhile this guy focuses on hips

//curves, which I hate, die curves die
beginShape();
fill(#FFFFFF);
// left curve
vertex(leftCurveStartX, leftCurveStartY); //sets the left shoulder
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point
curveVertex(leftCurveStartX, leftCurveStartY); //first actual point
curveVertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //hip
vertex(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5); //guiding point

// right curve
vertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //right bottom point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //guiding point
curveVertex(rightCurveStartX-innerCurve, rightCurveStartY+torsoLength*.5); //hip
curveVertex(rightCurveStartX, rightCurveStartY); //top right point
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
vertex(rightCurveStartX, rightCurveStartY); //sets the right shoulder
//give the shoulders a small amount of curve...hopefully
curveVertex(rightCurveStartX, rightCurveStartY); //guiding point
curveVertex(rightCurveStartX, rightCurveStartY);
curveVertex(brainX, rightCurveStartY-2); //puts a touch of curve, making the shoulders
curveVertex(leftCurveStartX, leftCurveStartY);
curveVertex(leftCurveStartX, leftCurveStartY); //guiding point

vertex(leftCurveStartX, leftCurveStartY); //finishes the shape using the first point

dendShape();

//bottom half
rectMode(CORNER);
rect(leftCurveStartX+innerCurve, leftCurveStartY+torsoLength*.5, shoulders-innerCurve*2, bottomHalf);
fill(shade);

//hands
ellipse(leftHandX, leftHandY, handSize, handSize);
ellipse(rightHandX, rightHandY, handSize, handSize);

//feet
ellipse(leftFootX, leftFootY, footSize, footSize);
ellipse(rightFootX, rightFootY, footSize, footSize);

//head
ellipse(brainX, brainY, diameterX, diameterY); //ccl head

fill(shade);
}
}

Philip

PImage img1, img2;
void setup() {
    size(600,600);
    background(0);
    img1 = loadImage("1_tracy.png");
    img2 = loadImage("2_tracy.png");
}

void draw() {
    fill(100);
    ellipse(350,500,50,50);
    ellipse(250,500,50,50);
    ellipse(400,280,50,50);
    ellipse(200,280,50,50);
    rect(280,300,40,100);
    rect(285,100,30,100);
    quad(220, 200, 380, 200, 320, 340, 280, 340);
    ellipse(300,130,100,100);

    image(img1, 100, 50);
    image(img2, 400, 50);}

APPENDIX C: FIGURES AND IMAGES

Unless otherwise noted, the figures were created by the writer for academic purposes.

1. This image shows an example of the LOGO interface and its coding language. Image courtesy of Wikipedia Commons.


2. This is an example screen taken when interacting with the Code Academy website. This image displays the specificity required of using Code Academy.

Screen capture of: http://www.codecademy.com/courses/getting-started-v2/0/3#
3. This is another example screen taken while interacting with the Code Academy website. It displays an error due to the interface.

Screen capture of: http://www.codecademy.com/courses/getting-started-v2/1/3#

4. This is another example screen taken while interacting with the Code Academy website. It displays an error due to an issue with capitalization.

Screen capture of: http://www.codecademy.com/courses/javascript-beginner-en-BtheymskY8/0/2?curriculum_id=506324b3a7dffb00020bf661
5. A screen shot of *The Sims 3*. Creating your Sims is an essential part of gameplay.

6. A screen shot of *Second Life*, the online synthetic world. Note the variety of human avatars.

A screenshot of *Whyville*, a science learning environment. Avatars take up a significant amount of participant activity.


This screenshot depicts avatar building in *World of Warcraft.*
9. Initial screen for the Paper Doll Project

Hi guys! Meet Ada.

Ada is made using Object-Oriented Programming, or OOP.

Press M to continue.

10. Instantiation screen from the initial Paper Doll Project

Objects work like complex custom functions. You can pass arguments and create multiple variants. Make many Adas using the button.

Press M to continue.
11. Variable screen from the initial Paper Doll Project

Each object behaves on its own. For example, watch these Adas. Change the variables on one. It won’t affect the other.

Press M to continue.


Objects can also execute methods. Each object executes customized functions without affecting others. Experiment with methods using these buttons. The doll will only execute what is ‘on.’

To the Left
To the Right
Now Kick

Press M to continue.
Screenshot of Rose’s final project. Her project focused on aesthetic changes. She tried to make an avatar somewhat like herself.

Eric’s Admiral Ackbar code. He chose to focus on an outside narrative and fantasy.
Belle’s avatar with its first expression.

Belle’s avatar with its second expression.
Belle’s avatar with its third expression.

Belle’s avatar with its fourth expression.
Philip’s mockup. Note how he completely removed the avatar from its original program, in order to allow for new methods.

Ariel’s remixed cat avatar.
Flynn’s pirate avatar. Flynn was the only one to transform the curve function that draws the upper body.
BIBLIOGRAPHY


