VISUAL CODE

BREAKING THE BINARY

INTRODUCTION

“DECODING CODING”

Access to introductory code and programming education remains incredibly low. The computer, once touted as "the Great Equalizer" and "bicycle for the mind" remains fully available only to the few.

Yet there are countless innovations yet to be made in the field of computing. This is startling, considering how much change it has already made in our lives. In a world driven by computing, programming and computer science should be accessible to everyone.

Current solutions are highly skewed towards more "scientific" thinkers. In our interviews with over 400 people, this theme of their experiences became quickly evident. An overwhelming number of people shared that they were turned away from code and programming within their first hour of experience with the topic. Consistently, people related that they quickly hit a “wall of math and algorithms”. Alarmingly, many had such a negative experience that they felt they would never revisit coding entirely. Currently, there is a huge emphasis on teaching code and programming across the United States, often in the context of the "Hour of Code" program, in which young people are introduced to the topic in just an hour. If these programs are failing an overwhelming percentage of students, however, alternative methods of introducing the subject become necessary.

This project seeks to create an accessible programming language that is more visually based. Although some solutions exist, namely MIT’s Scratch, nothing has caught up to the mobile age. This project aims to reframe creating a game or app into the context of telling a story, putting character creation first. By researching story-telling and how people learn, and by applying technical and user interface design knowledge, this project intends to deliver a software solution that opens introductory coding education to more people.

ONLY

1/4

OF U.S. SCHOOLS

OFFER COMPUTER SCIENCE CLASSES

CODE.ORG

OUR TEAM

Jacob Johannesen
Design

Andrew Adriance
Development

SECTION 2

RESEARCH

Inaccessibility In Computer Science: A Longstanding Tradition

Access to introductory code and programming education remains incredibly low. The computer, once touted as "the Great Equalizer" and "bicycle for the mind" remains fully available only to the few.

Yet there are countless innovations yet to be made in the field of computing. This is startling, considering how much change it has already made in our lives. In a world driven by computing, programming and computer science should be accessible to everyone.

Current solutions are highly skewed towards more "scientific" thinkers. In our interviews with over 400 people, this theme of their experiences became quickly evident. An overwhelming number of people shared that they were turned away from code and programming within their first hour of experience with the topic. Consistently, people related that they quickly hit a “wall of math and algorithms”. Alarmingly, many had such a negative experience that they felt they would never revisit coding entirely. Currently, there is a huge emphasis on teaching code and programming across the United States, often in the context of the "Hour of Code" program, in which young people are introduced to the topic in just an hour. If these programs are failing an overwhelming percentage of students, however, alternative methods of introducing the subject become necessary.

This project seeks to create an accessible programming language that is more visually based. Although some solutions exist, namely MIT’s Scratch, nothing has caught up to the mobile age. This project aims to reframe creating a game or app into the context of telling a story, putting character creation first. By researching story-telling and how people learn, and by applying technical and user interface design knowledge, this project intends to deliver a software solution that opens introductory coding education to more people.

Seymour Papert's Mindstorms: Looking Back 36 Years

Seymour Papert has been a leading figure in computer science education for young people, and practically invented even the notion of it. He has spent this greater part of his life on this issue, and wrote Mindstorms: Children, Computers, and Powerful Ideas in 1980. He also helped to create the Logo educational programming language. Is his own words:

> *In most contemporary educational situations where children come into contact with computers the computer is used to put children through their paces, to provide exercises of an appropriate level of difficulty, to provide feedback, and to dispense information. The computer programming the child. In the LOGO environment the relationship is reversed: The child, even at preschool ages, is in control: The child programs the computer.*

Papert builds on this capability of computers as a technology, emphasizing the importance of the platform's interactive nature. Beyond this, he even relates computers to the advent of the printing press--a technology with the potential to completely change information, but this time not only in the way we communicate that information, but also in the way we create and produce it.

With speaking of such change, Papert is quick to mention the potential drawbacks of the post-computer society. Papert mentions the techno-Utopian critics of 1980, who fear that the "holding power" and psychological effects of computers can be incredibly detrimental to to mind. 1980's fears even include "students spending sleepless nights riveted to the computer terminal, coming to neglect both studies and social contact". In 2016, this fear continues and has essentially been proven true. Technologies that were unforeseeable in 1980, including social media and engrossing video games, are a huge source of addiction. A recent report by Common Sense Media has found that teens spend an average of six hours a day using social media. Ultimately, teenagers of 2015 (year of study) represent one of the first generations to grow up having complete access to the internet and social media, making these findings especially troubling.

Papert, however, in fact agreed with the critics of his time, maintaining that while the negatives were real, it was important to focus on and develop the computer's positive potential.

Logo: A Virtual Material

Seymour Papert worked with the MIT Media Lab to develop the LOGO programming language. He believed that children learn best when given material with which to create new things. LOGO is based on the idea of a “turtle”, a simple and basic component that can be used in various combinations to create more complex systems—similar to conventional programming languages. Nonetheless, LOGO is very easy for anyone to simply pick up and start using. It is based on simple movement commands—move forward, rotate to the right, etc. By chaining these basic commands together, LOGO users can instruct the turtle to draw pictures. LOGO continues to have huge influence today, with recently developed introductory coding programs including Hopscotch that use similar principles.

Scratch: Puzzle Pieces

In 2002 Mitchel Resnick introduced Scratch—a visual programming language based on snapping puzzle-like colored blocks together. This project, based upon Papert’s research, also came out of MIT’s Media Lab. Scratch allows for more complicated programs and games to be built than what was possible with LOGO, and the system that it established continues to be a standard in introductory programming education today.

An Aging Standard

Despite being created almost 14 years ago, Scratch’s influence remains strong in introductory programming.

A Standard That Hurts Students

In 2016, the world of technology is incredibly different than that of 2002. Despite this, there have been minimal efforts to create new introductory system to engage students. Instead efforts, such as code.org, have concentrated on pushing existing technology through programs like Hour of Code. Through our interviews with over 200 educators, parents, and students we have found that while Scratch helps engage more students, a significant percentage of students are still turned away from code and programming within the first hour of introduction to the subject. Ultimately, while efforts to increase computer science education at the organization level are admirable, we also must focus on technology solutions that can engage different types of learners.

Additionally troubling is the concern that STEM-focused education overlooks, and in many cases even devalues the arts. People that we talked to brought this pain point up repeatedly. In our own solution, we seek to bridge the “divide” between art and computer science by incorporating the arts throughout.

SECTION 3

DESIGN

Swift-Based iOS App

For this project, we decided to build our software solution for the iOS platform. This is because iOS-based iPads are commonly used in primary education settings. We also considered developing for Google Chrome—a platform that has quickly grown in popularity within schools due to low pricing. Ultimately, however, we found through interviews that students, parents, and educators alike had incredibly negative experiences with the platform. Beyond this, mobile devices have an unparalleled ability to engage younger audiences.

Overall Flow

Ease of use is mission critical to our software solution. Our app has to be easy to navigate and understand. When researching competing apps, we consistently found context issues. Sub-menus of these apps built up and up, obscuring their purpose.

For our software solution, we knew that our goal of having users build their own games and apps made it incredibly tempting to build up similar layers of complexity. Instead, we decided to break down the navigation into easy to understand and universal parts.

Navigation Bar

The navigation bar allows users to move between the different parts that make up their app or game. We have defined these three sections as objects, code, and scene.

Objects
This section is where users can create new objects. Objects are the first section for a reason—as both the code and scene sections build upon and use these objects. Beyond this, interviews and research identified objects (or characters) as the starting point when someone without coding experience thought about building a game.

Code
This section is where users can add behaviors to objects that they created in 'Objects'.

Scene
The scene allows users to place the objects they have created into a world.

Objects Bar

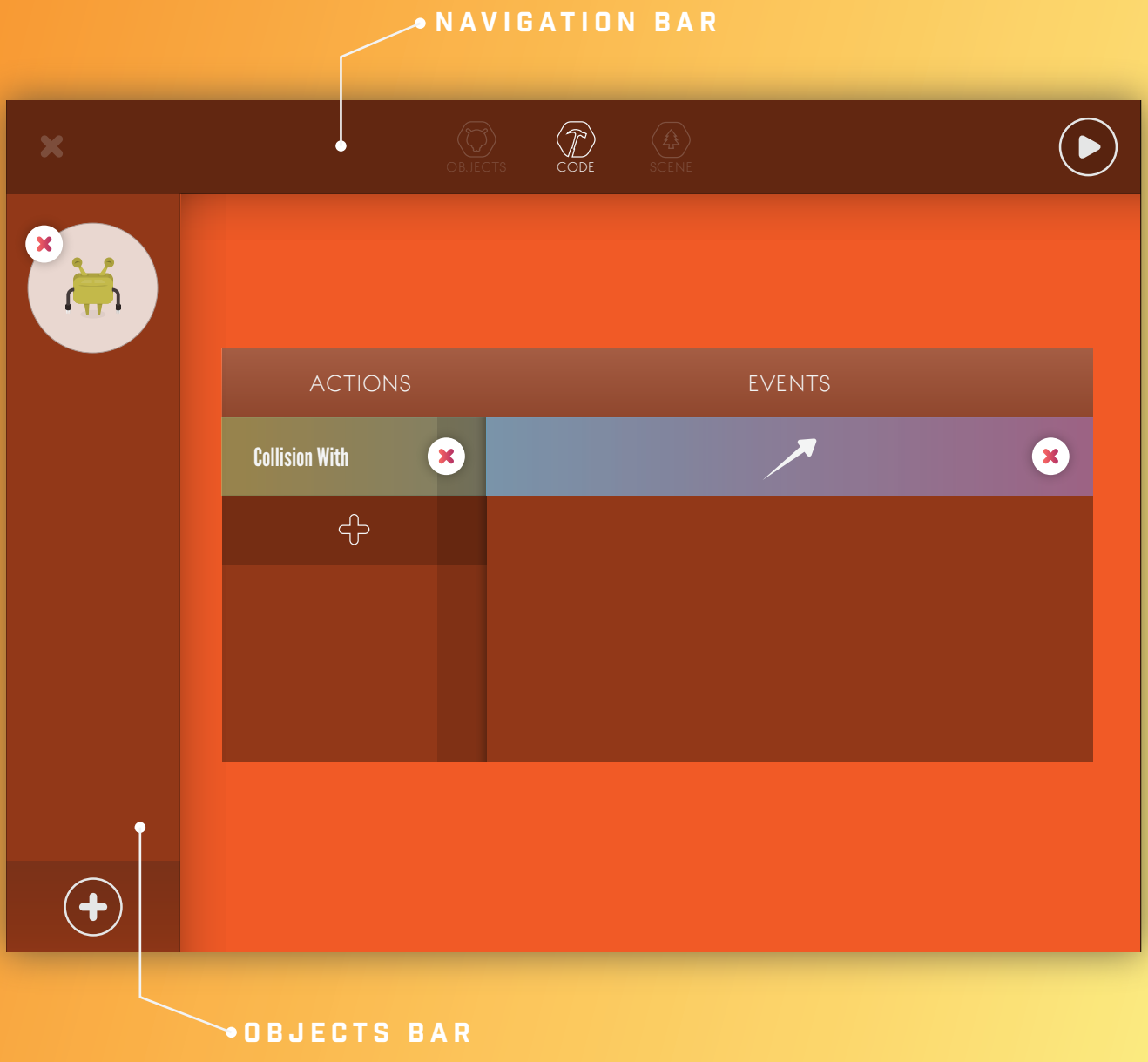
The objects bar is located on the left on the screen, and importantly is persistent whether users are currently in the objects, code, or scene section. All sections relate back to objects, and so access to them is always upfront.

Widgets

Widgets pop up when a user wants to do some more advanced customization. Widgets interrupt the app, blurring the previous context and creating a new one on top. Widgets cannot link to anything else. With only one level of abstraction at a time possible, widgets seek to make coding that first app or game less confusing.

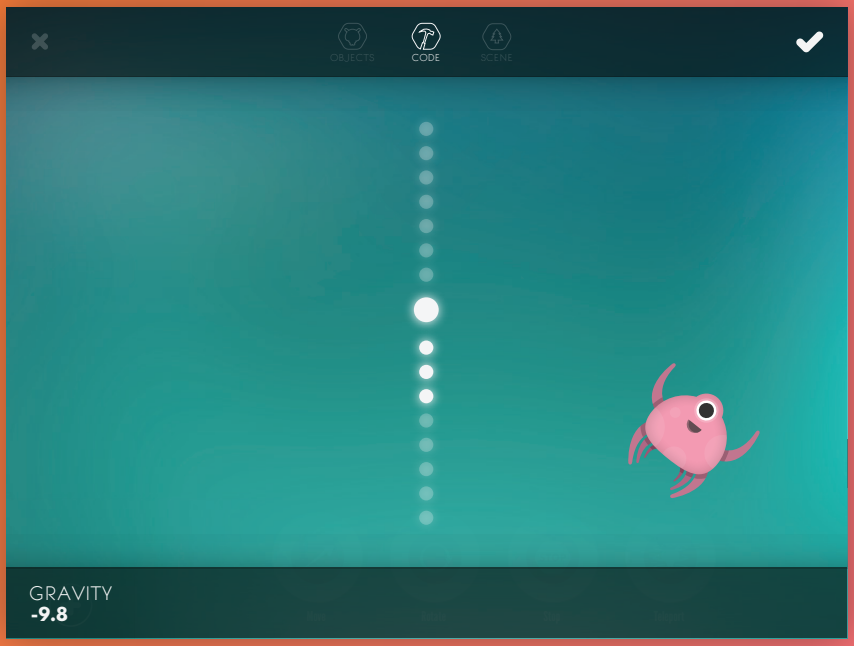
Building from Components, Not Templates

A common complaint among interviewees is that Scratch-based solutions only offer stock images to use in games. Beyond this, some solutions offer very primitive templates from which to create characters. Drawing inspiration from LEGO bricks and Minecraft, we have decided to take a more modular approach. In our design, component pieces can be combined in a variety of ways to create a huge variety of objects. In his writings on how kids learn with computers, Seymour Papert constantly relates back to the idea of “materials” that can be shaped built upon. For our software, we want to make sure that even the graphics were based on what the user wanted to make. No templates or pre-made junk here.


NAVIGATION BAR
OBJECTS
CODE
SCENE
ACTIONS
EVENTS
Collision With
OBJECTS BAR

WIDGETS

SET GRAVITY



SET TIMER

BUILDER

Uses components, similar to LEGO bricks.

SECTION 4

PERSONAS

Name
Matthew Murillo

Age
18

City
Gilroy, California

Access to Computer Science Classes
Low

Description
Matthew is a senior at Gilroy High School. He excels at school, but is worried about starting a computer science degree at Berkeley in the Fall after not passing the AP Computer Science exam. He has a passion for computers, but his school simply doesn't offer computer science classes—so the knowledge he does have, he learned on his own

This year, Matt has even collaborated with friends on an app they were planning to build. The idea was a space-themed game, and the group used Google Drive to share story lines and sketches. When it came time to actually build the game, however, their plans fell flat with a short-lived journey into coding and programming apps.

As he remembers the experience he wonders if maybe computer science just isn't for him.

Name
Melissa Benjamin

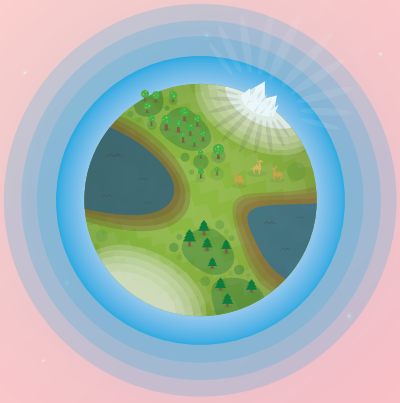
Age
12

City
Bellevue, Washington

Access to Computer Science Classes
Very High

Description
Melissa started coding in elementary school, and has a real talent for it. She makes simple apps in her free time using Scratch, which she learned at her STEM-focused and very well funded school.

Now that Melissa's in middle school, however, she's starting to notice some things. When encountering a problem in coding, Melissa visualizes herself inside the computer and talks about the iPad as if it were a person. When she communicates this way of thinking to her peers, however, they laugh at her and call her wrong. This despite the fact that she consistently does better than them with tough problems in code. Melissa decides it's easier to simply not explain her coding problems in the way that she actually thinks about them for now.



SECTION 5

CONCLUDING THOUGHTS

Analysis and Verification of Success

For this project, success will be measured by ability of users to complete certain tasks. In some cases, this may be on a scale of 1 - 10 based on how difficult or easy said task was to complete. These measurements will be taken with subjects who use both our software solution and existing software. We will then be able to assess the percentage difference in ease of use of our software.

Beyond this we will also be measuring user engagement, and perception of code and programming based on their experience with our software.

Interviews

In interviews with over 200 educators, parents, and students we attempted to narrow down specific concerns that resonated with people. A top concern among those we spoke to was “a wall of algorithms, math and code”, “coding being boring”, and “being unable to express creativity on the computer”.

Societal Impacts

Already, research shows the field of computer science to be inaccessible to many. Nonetheless, our society continues to push young people towards the fields—while not attempting to understand the issue of why so many are turned away in the first place. Ultimately, this project aims to build a more accessible introduction to code and programming.

Future Work and Next Steps

For this project, we plan to eventually release our software solution as a consumer product.

Conclusion

In 2016, information and technology dominates our lives—with data being collected on every aspect of the way we live our lives and screens becoming an extension of our own body and mind. While our use of technology has increased exponentially, our ability to program computers has simply not caught up. This is not only economically devastating, with technology companies unable to fill positions, but also culturally problematic. The computer, as described by Seymour Papert, is an incredible tool because of it's bidirectional nature—it can talk to you and, in theory, you can talk back though code and programs.

Beyond this, a certain culture has developed around computer science and programming. This culture has created a mythology of programmers as "wizards" and "rock stars", terminology used in many Silicon Valley job descriptions. Sprawling corporate campuses claim to shelter "top talent" as they "lead the world" though "innovations" that in reality are little more than a phone with a slightly improved camera or 10% more processing power.

Ultimately, Papert's vision of the computer as a tool for everyone has not been realized. In his writings, he compares the computer to the advent of the printing press. Today, we are faced with the question of whether we will promote computer literacy for all or continue to reinforce the "Silicon Valley" mentality of computing for the few.

WORKS CITED

Works Cited

"Every Child Deserves Opportunity." *Code.org*. Code.org, n.d. Web. 8 May 2016.

Papert, Seymour. Mindstorms: Children, Computers, and Powerful Ideas. New York: Basic, 1993. Print.

"Scratch - Imagine, Program, Share." *Scratch Help*. MIT Media Lab, n.d. Web. 8 May 2016.

Turkle, Sherry, and Seymour Papert. "Epistemological Pluralism: Styles and Voices within the Computer Culture." *Signs: Journal of Women in Culture and Society* 16.1 (1990): 128-57. Web.