Visualizing Remixes in an Online Programming Community

by

Di Liu

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY September 2013

© Massachusetts Institute of Technology 2013. All rights reserved.
Visualizing Remixes in an Online Programming Community
by
Di Liu

Submitted to the Department of Electrical Engineering and Computer Science
on August 23, 2013, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Scratch is a block-based programming language and associated online community that allows children and young adults to learn to build interactive games and animations — and share their creations with one another. Scratch’s foremost goal is to inspire learning through exploration and creative thinking. A novel feature of its projects are their complete openness: a user can always open up another user’s project to see all of its code and assets, and begin tinkering with them as if they were their own. This new revision branches from the original project, creating what we call a “remix” project.

Recently, the authoring environment has been rebuilt as “Scratch 2.0”, which brings what used to be a downloadable file completely online. The community website is now integral, and remixing is as easy as clicking a button. This thesis documents the rethinking and implementation of the remix tree, a visualization which allows users to explore the branching structure of changes by different users to a project over time. Additionally, we analyze changes in usage behavior and user feedback. The result is a much more usable and visually appealing tree which handles massive data sets fairly well, but continues to require iteration.

Thesis Supervisor: Mitchel Resnick
Title: LEGO Papert Professor of Learning Research, Program Head in Media Arts and Sciences

This thesis is based on research supported by the National Science Foundation under grant numbers 1002713 and 1027848. Any opinions, findings, and conclusions, or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the National Science Foundation.
Acknowledgements

Thanks to my advisor, Mitchel Resnick, for inviting me to work with the Lifelong Kindergarten group, in a rather magical place where people never grow old. Thank you for being so patient with me, and teaching me that academia is not just about rigor—it can be personal and passionate. I hope you liked the shark.

Thanks to Sayamindu Dasgupta for fishing me out of storms and calmly fixing my disasters. You may not think much of it, but I, doubtless the whole Scratch community, couldn’t have made it far without you. Thank you also for writing NICE in very large letters on my various thesis drafts; somehow it really made all the hours worth it.

LLK has been an extraordinary group, full of some of the most creative, inspiring people I know, yet humble and always ready to help. Thanks to Eric Rosenbaum for his endless flow of awesome ideas, Champika Fernando and Glen Chiacchieri for guiding me through the labyrinths of Django and the rituals you have to do to get webpages to do what you want, John Maloney and Evelyn Eastmond for my fantastic UROPs with this group, Michelle Chung for inspiring me to believe, yes, you can be everything you’ve ever wanted to be, Abdulrahman Y. idlbi, Tiffany Tseng, Stephanie Gayle, Natalie Rusk, Philipp Schmidt, Shane Clements, and Mark Goff. You’ve all been like family.

Thank you to my friends, David Benjamin, Katherine Fang, and Joshua Alman, for always being there for me and guiding me through tough spots when they had their own theses to write.

And finally, thank you Mom and Dad, for teaching me how to fight for dreams.
Contents

1 Introduction .......................................................... 17
    1.1 Scratch ...................................................... 17
    1.2 Scratch 2.0 .................................................. 20
    1.3 Remixing ...................................................... 20
    1.4 Thesis overview ............................................. 24

2 Goals ................................................................. 27
    2.1 Basic functions .............................................. 27
    2.2 Usability ...................................................... 28
    2.3 Serve user goals ............................................. 29
    2.4 Encourage remixing ......................................... 30

3 Background .......................................................... 31
    3.1 The remix visualization ..................................... 31
    3.2 Other remix visualizations .................................. 34
    3.3 Track changes ................................................ 36
    3.4 This Exquisite forest ....................................... 36
    3.5 Other tree visualizations .................................... 39

4 Design Decisions ..................................................... 41
    4.1 Initial brainstorming ........................................ 41
    4.2 Choice of tree representation ................................ 47
    4.3 Project thumbnails ........................................... 48
        4.3.1 User feedback ........................................ 50
    4.4 Navigation ..................................................... 50
        4.4.1 User feedback ........................................ 52
    4.5 The classic problem with trees ................................ 54
        4.5.1 User feedback ........................................ 56
    4.6 Addressing hidden projects .................................. 56
        4.6.1 User feedback ........................................ 58
    4.7 Love-it and favorite indicators ............................. 59
        4.7.1 User feedback ........................................ 60
    4.8 Aggregate user feedback ..................................... 60

41
5 Implementation
5.1 Databases ............................................. 65
5.2 Libraries ........................................ 66
5.3 Splitting the model and view ......................... 67
5.4 Organization of the canvas .......................... 68
5.5 The tree layer ....................................... 69
  5.5.1 How do you avoid crossing branches? ........... 69
5.6 Pan and zoom layer ................................... 70
5.7 Page-turner layer .................................... 72
  5.7.1 Page-turn animation: Trouble with "rotate" .... 72
  5.7.2 Page turn animation: Accidental hovering ..... 73
5.8 Project thumbnails .................................... 73
5.9 Text layer ........................................... 73
5.10 Not showing hidden projects ........................ 74
5.11 Efficiency .......................................... 75
  5.11.1 Dealing with huge trees ....................... 76
  5.11.2 Text ........................................... 76
  5.11.3 Reloading thumbnails ......................... 77
5.12 Logging ............................................ 77

6 Analysis ................................................ 79
6.1 Timeline ............................................. 80
6.2 Comparison to previous remix tree .................. 81
6.3 Comparing to other remix displays ................. 82
6.4 Returning users ...................................... 83
6.5 Navigating the remix tree ........................... 84
  6.5.1 The average session ............................ 86

7 Assessment ............................................ 89
7.1 Basic functions ...................................... 89
7.2 Usability ........................................... 90
7.3 Serve user goals .................................... 92
7.4 Encourage remixing .................................. 93
7.5 Reflection ........................................... 94

8 Future work .......................................... 95
8.1 Short-term goals ..................................... 96
  8.1.1 Multiple views .................................. 96
  8.1.2 Branch colors ................................... 96
  8.1.3 Internationalization ............................. 97
  8.1.4 Navigation ...................................... 97
  8.1.5 Copy sharing same remix tree ................. 97
  8.1.6 Tree previews ................................... 98
8.1.7  Toggle hearts and stars .............................................. 99
8.1.8  Toggle hidden projects ............................................... 99
8.2  Long-term visions .......................................................... 99
8.2.1  Growing trees ............................................................ 99
8.2.2  Forests ................................................................. 101
8.2.3  User trajectories ....................................................... 103
8.2.4  Seasonal variation ..................................................... 104

Bibliography ................................................................. 106
List of Figures

1-1 The Scratch 2.0 authoring environment. At the top left, the user can see what the final product and animations will look like. A sprite selection pane is on the bottom left. The new backpack feature can be accessed along the bottom. Blocks from the palette down the middle can be dragged to the area to the right to give a set of instructions to the sprite currently showing. Blocks promote discovery by giving suggestions on how to begin, and resisting error by looking like they fit together only when they can be used together. The glowing set of blocks is currently being run by the editor, and so in the preview, the sprite is spinning. 19
(a) A typical sample of Scratch projects. 20
(b) Some of Scratch's most popular projects. 20

1-2 Scratch allows kids to create meaningful projects on par with the technology that surrounds them. In these examples, Scratchers have created an animated web series, a game that takes camera input, and an MMORPG where users can play in a world together real-time. 21
(a) The old Scratch website. 22
(b) The new Scratch website. 22

1-3 Code for any shared project can be viewed via the "See inside" button on the project page. 23

1-4 Once inside, any user can edit the project as if it were their own, with the "share" button being replaced by the "remix" button. 23

1-5 A user's backpack, where they can store and retrieve code and assets. 24

1-6 Examples of some projects specifically designed to be remixed. Some are designed for remixing art and some for remixing code; some are designed for many depth 1 remixes (e.g. art contests), some for a single long chain ("add yourself"s), and some are libraries for arbitrary remixing (the platformer base). 24

3-1 The remix tree before the release of Scratch 2.0. 32
3-2 Scratch sample projects are candidates for massive numbers of remixes. This tree is being capped at out-degree 300. 33
3-3 GitHub remix visualization. 35
3-4 This Exquisite Forest website entry page. 37
3-5 "The Forest". 37
3-6 An individual tree. This view is zoomed in to show the nodes. 38
Treevis.net, displaying a few tree visualization variants.

(a) A tree where each project node is represented by a thumbnail, and length of the branch between them (possibly curly) represents amount of change between pairs of projects. Time flows upward, with the longest chain being moved toward the center. Marks or hearts springing from project thumbnails give an idea of social activity around projects.

(b) Thumbnails representing the projects are connected by an edge if they're related. The network graph follows spring physics and attempts to distribute projects evenly. A popular project would stick out by being surrounded by a thick cluster.

(c) Scratch 2.0 introduces a new feature called the backpack, which allows projects to borrow individual chunks of code or media files from each other. This essentially creates multiple creditable parents. This tree tries to incorporate those parents. Once again, time flows upward, but now a project may have multiple parents, and the edge shows an icon for what was borrowed.

(d) Multiple possibilities are drawn. First, a traditional remix graph next to a player and with a description of the change. Then, several variants on trees, ranging from natural-looking to abstract, with time flowing horizontally or outward, and parents’ sizes depending on their number of children. The idea of a forest of trees. Finally, replacing the tree with a flower, where petals replace branches.

(e) Two ideas are drawn. In the middle, time is the x-axis and projects spawn off upwards, with length of the stem showing amount of change. Those projects in turn may have more children. The idea on the bottom attempts to tackle the idea of which thumbnails to show when there are too many. At any one time, one chain of projects (a leaf node and all its ancestors) has enlarged thumbnails.

(f) Expanding on the stress-one-chain idea. Users may change which thumbnails are enlarged by clicking on an unexpanded node, which enlarges all its ancestors and picks some chain of descendants (perhaps most popular?) to enlarge, giving the user a "suggested" path. Angle that the child comes off of its parent is related to time of remix, with leftmost being oldest. A project player is placed next to the visualization.

(g) A forest of related remix trees. Colors indicate borrowing from other trees.

(h) Modelled after the visualization for GitHub, time flows horizontally. Nodes are placed at share time of a project, but branching does not occur until later because fork time is at time remixing began. Hovering over nodes gives a pop-up containing a thumbnail and statistics for the given project.
(i) One of my designs. Only a few levels of depth of the tree are visible, and as the user hovers over a child project, its children surround it, and the user zooms into the new project. In case of too many child nodes, they stack up on the edges, and hovering over the stack brings a new set of children into view.

(j) Another one of my designs. The remix tree takes inspiration from actual trees, and each project is on a leaf. The group had talked about wanting to see how visualizations changed over time. A time slider along the bottom of the tree allows users to watch leaves sprout and the tree grow. Nodes on the time slider are project creation times and are snapped to.

4-3 The initial design that I settled on.

4-2 The design I thought we’d be releasing.

4-1 The colorful design I also tested, which we ended up releasing.

4-0 The glow, on a hovered and unhovered project thumbnail.

4-1 Small project thumbnails on each node of the tree, with a floating info box on one near the top that is being hovered over.

4-2 The original release on top versus the latest version on the bottom. Various changes are visible, such as the size increase in project boxes and decrease in child size drop-off rate in response to user feedback. (The “after” image is actually zoomed out more than the “before” image, due to the decreased drop-off making the tree wider).

4-3 Each child is a constant fraction of the size of its parent.

4-4 Image the above user submitted as an example of an unclickable node.

4-5 A fraction of the enormous mess that is in your browser’s memory.

4-6 Nested paginated branches. The two page turners have white branches, and look like a stack of projects.

4-7 This tree shows two hidden projects, one hovered and one unhovered. They are the projects showing gray Scratch cats, similar to the one you see when you navigate to a project that no longer exists. They do not link to any project pages.

4-8 A remix tree with a well-loved project at its center. The hearts spread outward in a flat oval to imitate tree leaves. Stars were later removed as being potentially too noisy.

5-1 A project with more spread-out children needs more angular space to avoid intersecting projects around it.

5-2 A Pythagoras Tree, an example of a tree in which the branches curl around to the extent that the tree self-intersects.

5-3 To avoid self-intersection from curling around, the spread of the children of any branch should be no more than 180 degrees.

(a) A tree that has been manufactured to have a profusion of hearts and stars. Even though each thumbnail is drawn after its own set of hearts and stars, hearts and stars from other thumbnails cover up previous ones.
6-1 Timeline of updates to the remix tree.
(a) Total pageviews over all remix trees for the lifespan new remix tree (2013 in orange), compared to statistics for the same period of the year for the previous remix tree (2012 and 2011 in red and blue).
(b) Average time spent on a visit to any remix tree for the lifespan of the new remix tree (2013 in orange) compared to statistics for the same period of the year for the previous remix tree (2012 and 2011 in red and blue).

6-2 All pageviews across the entire site for the same time periods.
6-3 Browsers used by remix tree visitors, vs visitors of any page of Scratch.
6-4 Methods users are using to see remixes. These are percentages of active users, users who have recently created and shared projects.
6-5 The frequency with which users of the remix tree have continued to use the remix tree. For instance, 13,163 IPs have visited the remix tree once and never returned, while a dedicated few have viewed thousands of remix trees over the last two months. Note that both axes are logarithmic in scale.
(a) For each session, the number of times a user has moused over an info box.
(b) The number of times a user has moused out of an info box.
(c) The number of times the user has clicked and dragged the tree to pan.
(d) The number of times the user has continuously moved the scroll wheel to change the zoom level of the remix tree.
(e) The number of times a user has pressed the zoom in button at the top left of the remix tree.
(f) The number of times a user has pressed the zoom out button at the top left of the remix tree.
(g) The number of times a user has pressed the zoom equals button at the top left of the remix tree, causing the tree to become zoomed all the way out. Note the change in scale for the frequency axis on this graph compared to the other graphs.
(h) The average number of times a given action was taken during a single session on the remix tree.
(i) The number of sessions in which the user used a particular method of scrolling, and how often they used both or neither.

7-1 The orange line is number of remixes created by month, and the blue line is number of root projects created by month.

8-1 The current button leading to the remix tree page. The tooltip appears on hover.
8-2 The forest from This Exquisite Forest, encouraging users to explore other trees.
8-3 In this example from Chapter 4, where donor projects appear near the tree, attached with special new edges indicating what was donated. This is one possible implementation of many.
Some sketches for possible trajectory designs.
List of Tables
1.1 **Scratch**

Scratch is a block-based programming language developed at the MIT Media Lab's Lifelong Kindergarten group under its head Mitchel Resnick. Resnick's advisor had been Seymour Papert, who defined the highly influential concept of constructionism: that people learn by building things. Papert said many subjects, especially math, are considered boring and even painful because they are taught using abstract concepts which students do not find meaningful. Rather learning should be a natural, exploratory, and exciting process, self-driven by personal desire to discover or construct something meaningful.
The kind of knowledge children most need is the knowledge that will help them get more knowledge ... which is why we need to develop a large range of mathematically rich activities or "microworlds." ... It is obvious that as a society we in the United States (and most places in the world) are mathematical under achievers. It is also obvious that instruction in mathematics is on the average rather poor. But it does not follow that the only route to better performance is the improvement of instruction. Another route goes via offering children truly interesting microworlds in which they can use mathematics as Brian did, or think about it as Debbie did, or play with it as Dawn did. If children really want to learn something, and have the opportunity to learn it in use, they do so even if the teaching is poor. For example many learn difficult video games with no professional teaching at all! [1, pp. 139-140]

In particular, Papert created the "turtle", a robot (in later iterations, a virtual character on a screen) that children could speak to using a mathematical language called LOGO (e.g. TURN 90, FORWARD 10), but whose actions, unlike math drills, manifested in reality. Even students with low to average performance in school discovered mathematical concepts by playing with this turtle. Many students soon improved in their other classes after discovering geometric and creative thinking concepts for themselves. Papert proved that more abstract subjects like math and computer science, once considered out of reach for all but specialists, were accessible with a change of approach. LOGO began and would inspire the field of children's programming languages, including Scratch.

Scratch, released in 2007, stays true to LOGO's spirit of learning complex concepts via discovery and creativity, while greatly expanding upon its capabilities. Scratch is a simple and intuitive programming language and graphics editor written for kids, yet powerful enough to develop games and applications nearly on par with those written by professionals. This allows kids to really get excited about what they create. As discussed in [2], Scratch's main contributions have been:

- Making programming more accessible. Scratch transforms commands into error-resistant drag-and-drop blocks; no need to remember commands or formats.
• Making programming more meaningful. Scratch adds a drawing tool and the ability to import a variety of media. Users can create immediately visible, sharable results.

• Adding a social community. More on this below.

Figure 1-1: The Scratch 2.0 authoring environment. At the top left, the user can see what the final product and animations will look like. A sprite selection pane is on the bottom left. The new backpack feature can be accessed along the bottom. Blocks from the palette down the middle can be dragged to the area to the right to give a set of instructions to the sprite currently showing. Blocks promote discovery by giving suggestions on how to begin, and resisting error by looking like they fit together only when they can be used together. The glowing set of blocks is currently being run by the editor, and so in the preview, the sprite is spinning.

Scratch has a thriving online community, scratch.mit.edu, where users can view, comment, tag, love, and remix one another’s work. The website had inspired over 300,000 users to share three million projects as of Dec 2012; about 1500 a day. The site got around 40k hits per day. That number jumped up to 80k with the launch of Scratch 2.0 in May of 2013.
CHAPTER 1. INTRODUCTION

1.2 Scratch 2.0

About 10 years after Scratch's conception, internet technologies were advancing rapidly and Scratch was looking like it needed a facelift. The authoring environment, which had been a downloadable file, was being brought online to make it easier for people to see and experiment with one another's projects. During the rewrite, code was optimized and many exciting features never possible before were added, such as the ability for projects to store persistent data, opening up the possibility of high score lists and multiplayer games. The Scratch website, which previous users didn't necessarily ever have to sign up for before, was suddenly the focus, and this demanded an overhaul of it as well.

1.3 Remiking

The concept of remixing is integral to the idea behind Scratch. In fact, the name "Scratch" comes from the term for turntable remixing of music. The idea of Scratch as a remixing community
Figure 1-2: Scratch allows kids to create meaningful projects on par with the technology that surrounds them. In these examples, Scratchers have created an animated web series, a game that takes camera input, and an MMORPG where users can play in a world together real-time.

is discussed at length in Designing for Remixing: Supporting an Online Community of Amateur Creators, [3] the Ph. D. thesis of Andrés Monroy-Hernández, one of the primary designers and engineers of the first version of the Scratch website.

Remixing provides many benefits:

- Remixing is foremost an avenue for learning. Anything interesting in the Scratch world can be cracked open and poked at. In theory at least, nothing is a black box, beyond the user’s control and reach.
CHAPTER 1. INTRODUCTION

The old Scratch website.

(b) The new Scratch website.
• Remixer provides frameworks. It takes less work to get from conception to vision by working off of one another. Learning to use frameworks—made easier by clean code compartmentalization—is an essential skill in computer science. Many projects are designed as templates for other projects, to be remixed. (This is expanded upon in Scratch 2.0 by the introduction of the backpack, which allows you to move chunks of code and assets from one project to another.)

• Remixer promotes collaboration.

• Remixer is a means for community projects; e.g. "Add yourself riding Nyancat".

Of course, remixer also has one strong disadvantage. Remixer makes plagiarism much easier. Accusations of stealing and not giving proper credit have been a big challenge for Scratch historically. Plagiarism is highly discouraging to hard-working Scratchers, and some users have pushed for allowing un-remixable projects from the get-go.

Figure 1-3: Code for any shared project can be viewed via the "See inside" button on the project page.

Figure 1-4: Once inside, any user can edit the project as if it were their own, with the "share" button being replaced by the "remix" button.
Figure 1-5: A user’s backpack, where they can store and retrieve code and assets.

Figure 1-6: Examples of some projects specifically designed to be remixed. Some are designed for remixing art and some for remixing code; some are designed for many depth 1 remixes (e.g. art contests), some for a single long chain (“add yourself”’s), and some are libraries for arbitrary remixing (the platformer base).

1.4 Thesis overview

In Chapter 2 of this thesis, Goals, I analyze what features the Scratch community wants and needs from the remix tree visualization. In Chapter 3, Background, I look at previous projects that have
had similar goals to find inspiration and a starting point for this one. In Chapter 4, Design Decisions, I reflect on the path of design decisions that brought the project to its current state, beginning with brainstorming around a table with fellow grad students in the lab, and ending with iterative improvement through integrating feedback from users. In Chapter 5, Implementation, I discuss the design of the backbone of this project: my choice of frameworks, code structuring, and some interesting challenges I encountered along the way. In Chapter 6, Analysis, I analyze another source of feedback that may reveal unconscious patterns and usability problems: the usage data for how users interact with remix trees. In Chapter 7, Assessment, I take to heart all of these results and feedback, and assess how well the project has lived up to its goals from Chapter 2. Finally, in Chapter 8, Future Work, I list future work: the many features I am still working on, and also some of the more farfetched, grander visions of what this project could look like and where it might lead next.
In order to understand what background is relevant for this project, we must first establish the purpose and requirements for a good remix tree. These goals will guide the building of and the eventual assessment of the success of the new remix tree. There are four main goals: basic functions (works and shows remix relationships), usability, serve user goals, and encourage remixing.

2.1 Basic functions

The remix tree needs to satisfy some conditions to be considered a remix tree.

Subgoals:
• Be able to see projects in the context of remixing. That is, if a “remix” of a parent project produces a child project, how a given project relates to its ancestors, siblings, and descendants.

It seems natural to represent this kind of relationship as a tree, however, remixing can be more complicated (e.g. cyclical collaboration, multiple sources). The visualization should accurately represent what is happening.

• Be able to easily browse projects that are related to one another.

• Be able to see useful details about projects.

• Be able to see differences between projects. If you’re browsing the remix tree, naturally you’re asking, how is the project changing over time?

• Be able to easily navigate to a remix tree that might be interesting, and from there easily navigate to a project that might be interesting.

2.2 Usability

First and foremost, the Scratch website is designed to foster a community around learning via discovery. Unlike many visualization tools, the remix tree is not designed for an internal group of trained users. It needs to be something practically any age group can pick up and use without explanation.

I'll use the 6.831 (User Interface Design) usability heuristics to judge usability. [4] They are an amalgamation and summary of Nielsen’s principles, Norman’s rules, Tognazzini’s principles, and Shneiderman’s golden rules.

Subgoals:

• Learnability
The remix tree is a means of discovery. It should be inviting (aesthetic), simple, and intuitive. When possible, the interface should be familiar (i.e. navigation is like Google Maps or Windows file explorer.)

- **Safety**

Safety shouldn't be a huge issue for a remix tree visualization, but is very important for avoiding fear of exploration. It says that the effects of any action that is not immediately reversible should be apparent, and that as many actions as possible should be reversible (e.g. you shouldn’t be able to delete projects on accident). Also, the webpage should be secure and not leak private information.

- **Efficiency**

The remix tree needs to be efficient enough to not detract from the user experience. This could be a problem for some projects with a huge number of remixes.

### 2.3 Serve user goals

There are some established things the community has come to use remixing and remix trees for, and the new remix tree should not make those things impossible. This is similar to but goes beyond the basic functions; it involves picturing representative target audience members and asking if the tree helps them meet their goals.

Possible use cases to consider:

- The host of an art competition might want to glance art changes over time.

- The writer of a game might want to glance at code/feature changes over time and incorporate them back into his game.

- A browser might just want to see something cool, or might want to find inspiration for their own project.
Chapter 2. Goals

- A remixer seeing a "Remix this!" project might be looking for the correct project to remix; possibly the latest, the best-written, or the one most similar to what they have in mind.

- An analytical thinker might just want to look for patterns in how remix trees grow.

2.4 Encourage remixing

The remix tree should encourage people to remix. We want the remix tree to be especially conducive to the good things that come out of remixing, like collaboration and learning, and discouraging to plagiarism.
3.1 The remix visualization

As the goal of the project is to update the remix tree visualization, the original visualization is a basic and important starting point.

The remix tree was developed as part of a Master’s thesis in 2010 by Rita Chen, titled “Scratch-Stats: A Site for Visualizing and Understanding Scratch Usage Data”. [5] It uses a modified version of the HyperTree visualization from the JavaScript InfoVis Toolkit, developed by Nicolas Garcia Belmonte. There are a daunting number of possibilities for tree representations. However, not much is said about the reason behind choosing that particular visualization library in the thesis,
except that “The visualization is fully interactive to promote exploration.”

All tree visualizations have to deal with the problem of fitting and accessibly displaying a potentially exponential number of child branches. In general, hyperbolic tree visualizations like the HyperTree are useful because they target this problem by stretching the space somewhat like a magnifying glass, compressing the edges so that the outermost branches get exponentially smaller. This particular visualization seems to further rectify the problem by dropping far away edges in chains that are too long, although these edges tend to behave incorrectly as you move toward them, making it unclear if the disappearing edges are a feature or a bug. The HyperTree allows each node to be labeled, and deals with what could otherwise be an unreadably thick label density at the edges by simply excluding labels that are more than two hops away from the center. The tree is navigated by clicking on labeled nodes, which then smoothly animates so that the chosen node is moved to the center.

The modifications Chen made were (1) to make the edges directed by adding arrows, since children need to be distinguished from their parents, and (2) to limit the number of children displayed per node to 300, since larger projects were freezing browsers. Remix trees this large are very rare, but
it is made up for by the fact that large trees are the most commonly viewed.

Figure 3-2: Scratch sample projects are candidates for massive numbers of remixes. This tree is being capped at out-degree 300.

In terms of the goals outlined in Chapter 2, this tree is actually pretty far from the mark.

1. Basic functions

We can see the structure of the tree, but the only information in the labels is the author names. Clicking on a node shows the project’s thumbnail, title, and author to the side. Thus at a glance, we can see how individuals behave, but not really how projects are related to each other. It’s easy to navigate to the remix tree, and the thumbnail links away from it. Cycles in trees are not allowed by the library, and so edges that would create a cycle are not added. However, the method for creating a cyclical dependency is somewhat convoluted, and it’s unclear what the correct way to represent this would be, so this solution seems sane.¹

¹It should not be possible to make cycles using the Scratch 2.0 editor. However, in the previous version Scratch 1.4,
2. Usability

The tree is simple and navigation is intuitive, aside from the restriction to only seeing locally. Some bugs detract from usability, and remix counts on the website are often incorrect, but these are reasonably fixable. Efficiency is only an issue for giant remix trees.

One usability issue that we'd like to fix is not easily fixable. The visualization was designed for undirected graphs. As a user navigates away from the root, the entire tree is transforming and the direction of arrows is shifting, so that it's non-trivial to figure out the relationship between any given pair of nodes, and becomes quite a challenge to find your way back to a node you were once at or keep track of where you are with respect to the overall structure of the tree. This isn't a compromising safety issue but can be very confusing.

3. Serve user goals

This remix tree is designed for the analytical thinker. It's not great for browsing projects, which was the focus of the other brainstormed cases.

4. Encourage remixing

This remix tree could encourage remixing, if a user saw a friend's name or wanted to create an interesting structure.

3.2 Other remix visualizations

Scratch is a peculiar case that mixes media and code, while also expecting its users to be non-technical. I would guess that no existing visualization library quite covers this use case. Freeform remixing of media-heavy projects certainly exists; for instance, of memes, music, and videos.

Visualizing changes to a project over time is a familiar problem in the world of version control. Git
(a method of version control) and GitHub provide graph visualizations, although these are built to track changes in text, and the merge operation means that they don’t form trees. [6]

![GitHub remix visualization.](image)

**Figure 3-3:** GitHub remix visualization.

Git deals with complicated changes that are otherwise not readily displayable pretty well: by requiring the user to provide a summary message for each change. These messages can be read by hovering over dots. Time flows along the x-axis, while users go down the y-axis. The color space separates “branches”, which are differentiated because they are explicitly named.

This kind of visualization could be good for Scratch projects. Right now the fact that one remix can continue to be edited and change over time isn’t acknowledged, but with Scratch 2.0 automatically saving a new version of your project every few minutes, this kind of versioning is now possible. The auto-saves don’t usually come at good stopping points, but there could be a manual save button that allows users to write in what changed.

One big difference is that there is one “master” branch in a version-controlled repository, which all branches want to eventually converge to. This means that Git visualizations will almost never be dealing with exponentially many branches. When Git does have too many branches to show comfortably, the inactive ones are pushed off of the visualization, so that’s a possibility.
3.3 Track changes

Microsoft Word, Adobe Acrobat, and other software for editing documents that must occasionally be collaborated on, have a feature that allows any changes to be highlighted to allow proof-reading by another party over email. Occasionally, users want to merge in changes that remixers made later down the chain, but Scratch right now has no intention of trying to support the hairy, confusing operation that is merge. Automatically highlighting a list of differences like this for each edge could be useful for Scratchers to figure out how much a project has changed and perhaps discourage would-be plagiarizers, at least for code, though it could look very messy and confusing on a crowded tree.

This kind of version control is starting to appear for media, too. PixelNovel allows Adobe Photoshop, Illustrator, or InDesign users to save and share changes to art files across SVN. Still, these image diffs aren’t quite correct for Scratch, which may contain animations or user-interaction-dependent behavior, and whose media is often made in other programs and imported rather than created in the Scratch editor.

3.4 This Exquisite forest

This Exquisite Forest is a website conceived by Chris Milk and Aaron Koblin, and produced by Tate Modern and Google. [7] It was designed as an experiment in collaborative creation. A user begins by drawing a short animation, and the website encourages users to write a few short rules, e.g. “Use only black, white, and yellow”, or, “Don’t destroy the box”. Anyone can then draw a few frames to continue the story, and the most interesting stories tend to grow into the longest branches.

“The Forest” is where users explore projects. Each tree’s starting animation is shown at its root. The structure and starting clip intrigue users to click through.

Once within a tree, a user can zoom and pan, hover over a leaf to play the animation from beginning
Figure 3-4: This Exquisite Forest website entry page.

Figure 3-5: “The Forest”.
Exquisite Forest, in my opinion, is beautiful and a lot of fun to explore. It uses the tree metaphor very cleanly to visualize a branching animation, with each segment doubling as a video player progress bar, allowing the user to see just which segment they’re watching. Exquisite Forest is very comparable to a Scratch remix tree in that the changes that it needs to represent may be visual or moving. It has it a little easier though, since the diff is always exactly the latest new segment of animation, whereas a project change in Scrach may affect the project from start to finish, or be invisible without just the right user inputs (say, fixing a corner-case bug). It also tends perhaps more to aesthetic than we’d like, allowing each new segment of the tree to grow naturally, which may cause densely intersecting branches that become hard to mouse over. Nonetheless its design an aesthetic are a big inspiration.
3.5 **Other tree visualizations**

Because of how families of remixes come about, they are very naturally represented by a tree structure. If we take a step back and simply look at various visualizations for trees, the possibilities are endless. Treevis.net cites a collection of 269 papers on different tree representations. [8].

![Treevis.net](image)

**Figure 3-7:** Treevis.net, displaying a few tree visualization variants.

For the sake of brevity, let’s just say that most of them are artistically inclined (more for the big picture than readability of individual nodes) and hierarchical, emphasizing parents much more than their children, or showing few children, rather than made for emphasizing diffs.
4.1 Initial brainstorming

My general sense from talking to people about what they wanted from the remix tree and looking at existing visualization libraries was that the Scratch remix tree was a unique case that required a new approach. And so, rather than finding an existing library, source-diving it and highly customizing it, I decided it would be about as efficient and effective to start from scratch.

Early in the process, I met with my fellow grad students in the Lifelong Kindergarten research group for a quick 15-minute brainstorming session where I asked them to sketch out what they thought the ideal remix tree visualization would look like. The sketches below are credited to
(a) A tree where each project node is represented by a thumbnail, and length of the branch between them (possibly curly) represents amount of change between pairs of projects. Time flows upward, with the longest chain being moved toward the center. Marks or hearts springing from project thumbnails give an idea of social activity around projects.

(b) Thumbnails representing the projects are connected by an edge if they’re related. The network graph follows spring physics and attempts to distribute projects evenly. A popular project would stick out by being surrounded by a thick cluster.

them: Champika Fernando, Tiffany Tseng, Eric Rosenbaum, Ricarose Roque, Abdulrahman Idlbi, and Sayamindu Dasgupta.

In general, people wanted the remix tree to be a diff tool and a browsing tool. Suggestions tended to show image diffs via thumbnail, and code diffs via branch length. However, code diffs are difficult to quantify (how much has the project functionality actually changed?), and our project manager was concerned with users gaming the system. As a browsing tool, thumbnails, descriptions, and taking advantage of all aspects of the visualization (e.g. branch length, color) to convey more information came into play often. The general tone of the designs ranged from professional to wacky and friendly.
Scratch 2.0 introduces a new feature called the backpack, which allows projects to borrow individual chunks of code or media files from each other. This essentially creates multiple creditable parents. This tree tries to incorporate those parents. Once again, time flows upward, but now a project may have multiple parents, and the edge shows an icon for what was borrowed.

Multiple possibilities are drawn. First, a traditional remix graph next to a player and with a description of the change. Then, several variants on trees, ranging from natural-looking to abstract, with time flowing horizontally or outward, and parents’ sizes depending on their number of children. The idea of a forest of trees. Finally, replacing the tree with a flower, where petals replace branches.

Thinking back, a timeline like the GitHub visualization would have been more practical from a purely utilitarian perspective; it could show exactly when branches broke off, and perhaps show evidence of when a project had incorporated edits in a child remix back into the parent. It may be worth considering adding in the future as an alternate perspective. But I am a visual person, and I was captured early on by the idea of a single project blooming into a tree grown over time by the community. At the time, we were discussing a timeline feature, where the user could get a sense of the speed at which a remix tree had grown by dragging a slider. It felt like a substantial achievement to watch a living metaphor grow, rather than git branch lines zooming off to the right.

This led to me reading up on the role of aesthetics in interface design, and research on visualizations using living metaphors. Aesthetics make a user interface seem more usable, even if it isn’t. People are more likely to like, trust, and start using the interface, although eventually usability concerns dominate. [9] Meanwhile, living metaphors are a stronger motivator than visualizations
CHAPTER 4. DESIGN DECISIONS

(e) Two ideas are drawn. In the middle, time is the x-axis and projects spawn off upwards, with length of the stem showing amount of change. Those projects in turn may have more children. The idea on the bottom attempts to tackle the idea of which thumbnails to show when there are too many. At any one time, one chain of projects (a leaf node and all its ancestors) has enlarged thumbnails.

(f) Expanding on the stress-one-chain idea. Users may change which thumbnails are enlarged by clicking on an unexpanded node, which enlarges all its ancestors and picks some chain of descendants (perhaps most popular?) to enlarge, giving the user a "suggested" path. Angle that the child comes off of its parent is related to time of remix, with leftmost being oldest. A project player is placed next to the visualization.

(g) A forest of related remix trees. Colors indicate borrowing from other trees.

(h) Modelled after the visualization for GitHub, time flows horizontally. Nodes are placed at share time of a project, but branching does not occur until later because fork time is at time remixing began. Hovering over nodes gives a pop-up containing a thumbnail and statistics for the given project.
that simply look artistic, but they are a much stronger demotivator when used negatively; for instance, in an exercise app using fish health to show progress, users stop logging in at all when they know their fish are sick. So there were good arguments for using a living metaphor to draw in users, as long as it never showed signs of dying.

I met some resistance from my advisor, who was worried that going too far with the metaphor would detract from the simplicity and understandability of a simple graph. Enforcing this metaphor did provide a nice physical reference for interpreting symbolism (the “sky” changing color could represent the passage of time; greener branches could be interpreted as newer), but it also intro-
duced associations that might be interpreted incorrectly, and unnecessary restrictions (e.g. the way trees grow, the graph could no longer grow out in all directions, but had to grow mostly upward).

However, when I created a prototype for testing internally, with one UI skinned as a line graph, the other skinned as a tree (users were allowed to become familiar with the interface using the graph skin, which I thought I would be going with, then allowed to try the tree skin), everyone, including my advisor, liked the tree skin better. The group excitedly proposed ideas for changing seasons within the remix tree visualization and dangling Christmas lights during December. Admittedly I chose the pretty biased audience of a creativity and tinkering lab.

The remix tree ended up being released with a conservative but tree-like visualization. The tree was a black, rather unnatural color, the ground was undecorated, and the background only appeared when fully zoomed out. The idea was that the visual impact of the tree was useful when getting the big picture, but that it could be distracting when a user was trying to look at details.

![Figure 4-2: The design I thought we’d be releasing.](image)

Surprisingly, the biggest complaint initially in user feedback was that the tree didn’t look enough
Figure 4-1: The colorful design I also tested, which we ended up releasing.

like a real tree: as a forum poster said, "also what’s with the ground? the colors need a little work to make it more, you know, tree like." The design of the tree has eventually become more natural.

4.2 Choice of tree representation

We began by addressing the most apparent problem in the old remix tree: the constantly shifting view causing users to feel lost and lose track of the relationships between projects. We decided that rather than having a shifting graph, the tree would be a static thing, and the user would move around on it by zooming and panning.

However, another problem arose from this solution. A user’s “place” in reading along the tree, which used to always be the node that was dead center, was no longer being tracked, and so now users could get lost a different way. For this, we decided to add an orange glow to the focused project. When the user views the tree from a project page, the glow serves to distinguish the project that the user came in from, and as the user hovers over nodes in the tree, the glow moves
to the most recently hovered item, which I’ll call the “focused project”, and serves as a placeholder.

![Figure 4-0: The glow, on a hovered and unhovered project thumbnail.](image)

### 4.3 Project thumbnails

The desire to be able to see the projects in the remix tree from the remix tree was pretty clear from the onset, with almost all of the group’s brainstorm sketches involving thumbnails somewhere. There was the question of where to put them, and how large? We knew we wanted the focused project to display a thumbnail somewhere. We started by adding a large info box to the right of the remix tree visualization, which I will call the “info box”, and hovering over the nodes for projects in the tree would fill the info box with a large image, description, and links to the author page and project page. It was possible for the preview image in the info box to be an actual project player, as they are in the new site search feature. However, the remix tree didn’t feel like a place where users would sit and play through Scratch projects in turn, since by nature, the projects would be repetitive, so we went with just a static image.

It felt bothersome to constantly have to be looking off to the side and then back to the tree, and so the info box was moved to float under the cursor on hover. This box was designed to have a constant size (so that even if the user was zoomed out or in, the info box image and text would still be clear and readable.) The challenge with this change was that the info box couldn’t be too large, since it now blocked the view of the tree. Thus we had to dispense with showing the project
description (which we had hoped would help indicated what had changed in the remix) and now only show the title, author, and project image.

The question arose of whether we wanted these preview thumbnails to appear for more than just the focused project? Initially, we thought it might make the tree look too messy if every project’s node had a small preview image over it, and experimented with the idea of having only the focused project’s immediate parent and children, and perhaps siblings, sport preview images. In the end we found that having preview images over all of the projects looked very reasonable, as long as they weren’t too large. And so each node of the tree has a small preview image by default, with a large image and title text popping up over it on hover.

Figure 4-1: Small project thumbnails on each node of the tree, with a floating info box on one near the top that is being hovered over.
4.3.1 User feedback

Info boxes used to always grow to a constant size on hover. This meant that if the user was zoomed in on a project, hovering over the project thumbnail could actually cause it to shrink. One user reported this as a bug, although it was the intended behavior. It did feel weird so the hover was changed to not change the thumbnail size when it was already larger than the info box would be.

Initially, the thumbnails were 60 by 45 pixels and the thumbnail in the info box was 150 by 120 pixels. Due to feedback that the thumbnails were difficult to see, these were boosted to 68 by 51 and 272 by 204 respectively. The new larger info box size made it difficult to hover off. To help with navigation, which was hindered when the user needed to move the cursor a long way to hover off of the info box to make it disappear, the info box now disappears when the user hovers off of the size of the smaller thumbnail. This boundary isn’t actually visible when the info box is on top of it, but its actual area isn’t important, so much as users seeing that the info box disappears more quickly when moving the mouse.

4.4 Navigation

At first, the remix visualization was housed in a panel with two-way scroll. Users moved around the remix tree area by scrolling left and right, and could focus in on a project by clicking on the small preview of it at that node on the tree. In order to fit more projects within view and look more natural, each set of child branches was a constant fraction of the size of the parent branch. Clicking on a project to focus on it moved that project to the center and zoomed in on it so that no matter what project you clicked on, the resulting view and level of detail of children/ancestors would be the same.

However, it felt more natural to have clicking on projects move you to their project pages, so pan and zoom had to be implemented differently. The remix tree now follows the navigate model of Google Maps, where zoom is controlled by scrolling, and pan is controlled by click and drag. Zoom out is capped when you can see the whole tree, and users can zoom in indefinitely (our reason for
Figure 4-2: The original release on top versus the latest version on the bottom. Various changes are visible, such as the size increase in project boxes and decrease in child size drop-off rate in response to user feedback. (The "after" image is actually zoomed out more than the "before" image, due to the decreased drop-off making the tree wider)
4.4.1 User feedback

Initially, each branch was 70% of the size of its parent. This was fine because clicking on a project did the navigation for you, making it easy to see what was going on at the ends of chains just by clicking on them. However, since navigation has become manual, it has become tedious to view small projects. One forum poster says:

It seems that projects get small really fast, though: projects are almost unclickable after 6 or 7 remixes. This makes viewing trees for “add yourself” projects and other remix chains hard because you’re dragging the mouse all over the place.

I wouldn’t mind seeing less projects on each “page” if they were larger. Or maybe there
can be a “show more projects on this branch” button on long (5+ projects) chains so projects wouldn’t be so small.

In response, each branch is now 95% of the size of its parent and thumbnail size has been increased as stated in Subsection 4.3.1. This solution only mitigates the problem. Making the old style of navigation possible might be a better solution.

Figure 4-4: Image the above user submitted as an example of an unclickable node.
4.5 The classic problem with trees

The initial idea was to have branches squeeze tighter together when there were too many children, as in the old tree. Since a node with 300 children made the tree impossible to use anyway, this tree, which had to fit in preview images, would simply have a cut-off at a much lower number of children. Sadly, this meant that many projects would be inaccessible from trees simply because their predecessor was popular. Moreover, because they can be arbitrarily large, showing everything in the tree can be overwhelming and is usually a bad idea.

File system browsers often deal with this by having folders by default "closed", so that the sub-tree beneath is hidden until the user opens it. However, if a single node has too many children, a huge list to scroll through may result. Some programs simply cut it off.

We came up with the idea of paginating nodes with too many children. The concepts of tree visualization and pagination very rarely come into play together, which is odd because one is designed to handle the problem with the other, and caution states that we must take a moment to ponder why. There do exist a couple of examples of paginated trees. I found an untested example of one proposed for a file system browser [http://andreakendall.blogspot.com/2009/03/untested-idea-for-paginated-tree.html], and there is one for exploring heap space usage in a very hidden away corner of the Chrome developer console.

One thing that becomes apparent quickly when you actually try to use the heap browser is that the user experience is absolutely awful. The user is wading through megabytes of objects with tens of thousands of children each. Just scrolling on each level is already overwhelming. There is a search bar for specific queries, but if a user wants a general idea of the structure, they must perform a painstaking search. It is possible to sort on a number of different metrics, but the sort is universal. That is, if you would like to find the largest item that is a child of a given node, you must sort everything by size from every node, and can lose track of the node you were looking at. For these reasons, recent usability wisdom has been to avoid hierarchical tree structures and pagination entirely, and instead broadening search capabilities. Pagination is still useful when one
will probably find what they’re looking for within the first few pages, and for this reason search engines have continued to use pagination.

In our case, the tree is a change tree, not a hierarchy tree. A node’s descendants, ancestors, and siblings are about as important as its immediate parents and children. Since our remix tree is more geared toward overall structure than search (since Scratch has a separate search capability, and since relations between projects are not easily spelled out in words), it makes sense to have nodes at all levels by default “open”. This means that the only painfully large lists are in the paginated nodes of the tree. These continue to face all the same issues of hiding parts of the tree: they hide

Figure 4-5: A fraction of the enormous mess that is in your browser’s memory.
interesting structures, and only dedicated users will look past the first few pages. We considered adding ordering or filtering within each node, but this only partially addresses the issue while making the tree potentially much more complicated for new users. Hopefully, planned future iterations will be able to improve these issues.

Another possibility was to use continuous scroll rather than paginating; that is, rather than show in fixed blocks, the projects would continue to rotate through. This makes it somewhat more difficult to keep track of how far in you are, but feels more natural for something visual like images, which in our case is more important. However, pagination was chosen because it was much easier to implement, with the idea of switching to continuous scroll later on as needed.

To hint at their purpose, the page turners for the paginated nodes look similar to branches but faded out, and are in the location where you would expect more projects to be. The have a hover-over that shows the hand cursor and implies that there are more projects to be seen. The number hints at how much more of the tree the user hasn’t seen, and includes the count of all nested children, not just siblings.

4.5.1 User feedback

Despite us worrying that this would be a confusing mechanism, as it isn’t really seen anywhere else, we got no feedback about the pagination.

4.6 Addressing hidden projects

For various reasons, projects may be hidden.

- In the past, projects were shared by uploading them, and the local copy on the person’s computer could continue to be modified without affecting the upload. Now that all project editing is online, editing a shared project could break it for others viewing it, or mess up the state of persisted data. Therefore to take projects off the air, there is the new concept of
Figure 4-6: Nested paginated branches. The two page turners have white branches, and look like a stack of projects.

"unsharing" a project. These projects shouldn’t appear the same as other projects in a remix tree, since the user clicking through will be unable to see the project or comments for an unshared project that isn’t his or her own.

- A user may delete their own project.

- A project might get censored by a moderator for inappropriate content or language. It’s a kids’ site, after all.

- A project may get removed to enforce copyright.

Is it interesting to see projects that have since been deleted or unshared? Several Scratch admins thought so, as it was interesting to see what projects inspired deleted projects. At first these were left in, with their project image replaced with a gray Scratch cat and “No longer available” in the title line. However, there were so many of them that it cluttered trees and left them with pages
CHAPTER 4. DESIGN DECISIONS

Figure 4-7: This tree shows two hidden projects, one hovered and one unhovered. They are the projects showing gray Scratch cats, similar to the one you see when you navigate to a project that no longer exists. They do not link to any project pages.

and pages of no real projects. (This was partially due to a bug that caused newly created projects to automatically become shared. New projects were created simply by going to the Scratch editor web page, due to autosave.) Since they seemed to be a hindrance, these nodes were removed entirely from the trees. However, in some cases, a project was remixed before it was deleted or unshared, which left the tree in a strange disconnected state. In this case, the gray box still appears as a placeholder, even though such projects don’t appear if they are leaves.

4.6.1 User feedback

Several users said that they didn’t think unshared projects should be in the trees, or that it was confusing that there were unshared projects that were end nodes. After they were removed, one user said that they would like to have the option of making them visible.
4.7 Love-it and favorite indicators

Figure 4-8: A remix tree with a well-loved project at its center. The hearts spread outward in a flat oval to imitate tree leaves. Stars were later removed as being potentially too noisy.

The scrollable branches are sorted on time first shared. This means that by default, users are looking at oldest projects first. This makes sense for many remixes where users are trying to build a long chain ("Add yourself" projects).

However, many other uses of the remix tree, such as finding inspiration and finding a good project to learn from, would get better use out of the remix tree if it sorted based on popularity. However, since we didn’t want to implement node sorting for individual nodes, and setting sort overall
causes a rash of problems, we would have to represent popularity in a different way. There is something called a "love-it" on the Scratch website with a well-recognized symbol of a heart that allows users to show their love for projects. These are displayed on the tree as hearts behind the project that is loved. Since it's meant to simply give an at-a-glance indication of what might be interesting, there is a cap to the number of hearts so that they don't slow down performance.

The Scratch website also has something called "favoriting", also using a well-recognized symbol. However, favorites and love-its serve similar functions in terms of determining popularity, and so we decided to show only the hearts for simplicity.

### 4.7.1 User feedback

Two users suggested that we add stars for favorites after we added hearts for love-its. I asked for an argument as to why they were different from love-its as indicators, and didn’t hear back.

We were worried about even just the hearts making it difficult to focus on the projects in the tree, and initially planned to have a button to toggle their visibility, but there has been no negative feedback about hearts.

### 4.8 Aggregate user feedback

Feedback was gathered via the Announcements forum, where users were prompted simply with three bullets:

- What do you like about the new tree?
- What do you think we can improve?
- Report any bugs you find here!

While this thread continues to aggregate feedback, as of August 15, 2013, the topic has 1705 views (counting returning users) and 57 replies. Here are numbers giving a feel for strength of opinions
rising from the thread. My responses aren’t included, and multiple posts within a conversation about a given feature were counted as one opinion. A single post may have fallen under multiple categories. Posts that weren’t relevant (e.g. “How Did You Post in Announcements? You’re Just A Scratcher!”) and posts that were very difficult to understand (e.g. “that is remix tree on this project was too many remixes?”) were excluded. The views may be misrepresentative because the same users tend to dominate activity on the forums. 25 different users responded.

- 8 posts are about bug reports, 3 about how the starter project Pong, with over 14 thousand descendants, froze computers. This one and one other about author name showing up as ‘undefined’ was addressed. 1 bug about copied projects not appearing on the remix tree needs to be addressed. 1 bug was that you could tell that an unshared project existed if you guessed its ID number (it gives a tree with a single, grayed-out node), but we didn’t bother to resolve this. 1 bug was that tree branches were dipping into the ground, a somewhat rare case which has yet to be addressed. Another bug was due to the data structure on the server being incorrect. A few such instances could be found around the Scratch website (projects report their parent or root wrong, cutting off the tree or making it empty). It seems like they might be caused by network blips or bugs in upload code.

- 6 users said it did not look enough like a tree or made suggestions for things that would make it look more like a tree. In particular, people complained about the gray color of the branches, lack of leaves, and empty ground. In response, the color of the tree was tweaked to be slightly more brown, and the ground was rounded. Trees with a lot of love-its have hearts suggestive of leaves, but making the aesthetic too elaborate seems contrary to the goals of the remix tree.

- 5 users (mostly early on) said they liked that the look of the new tree and that it looks like something from nature.

- 3 users didn’t like the fact that projects which had been unshared (and were now gray) were still in the tree. Once they were removed 1 user said that they wanted to be able to toggle whether they could see them, just to get a sense of how large the tree was
• 3 users liked the idea of the project you had come in from being at the “ground” of the tree, everything below it being rendered as “roots” and everything above it looking like the usual tree. In particular, one user suggested it and 2 others chimed in with approval. This sounds pretty cool, but changing a tree graph in this way would make the directionality of parents and the location of the root very difficult to see at a glance. Additionally, the tree would change drastically for each different node that the user entered from, potentially making it easy to lose track of where you are in the grand scheme as was a problem with the original remix tree.

• 2 users misinterpreted the number shown on the remix tree button on the project page and reported it as a bug. We decided that this number should be the total number of remixes in the entire tree, as that information is interesting for someone exploring remixes, and isn’t available elsewhere. Other forum users corrected them, but it’s unclear what that number means. In the old Scratch website it was the number of descendants of this particular project, so it’s confusing for users to see ‘1’ next to a project with no remixes, or a large number next to a project with no remixes, but that is part of a large tree.

• 2 user thought we should add stars for favorites after we added hearts for love-its.

• 1 user liked that deleted projects in trees were now indicated.

• 1 user thought the new tree was “much more navigable”.

• 1 user (and many more in the weekly announcement thread introducing the feature and many others) was happy that the tree was back.

• 1 user didn’t mind if the tree was less dense so that it could show project thumbnails larger.

• 1 user thought the rendering as a tree was “a bit forced (thanks to the limits on projects in the tree that are on-screen at a time). It would have been better to just work off of the old design.”

• 1 user complained about performance on mobile devices.
Overall, I would have liked a larger sample size, but we were able to make some good improvements thanks to user feedback.
5.1 Databases

For Scratch 2.0, all of the databases were being cleaned out and rewritten, and that included the Remix tree database. The previous one had been in MySQL. This meant that each time a remix tree was requested, the data for the tree had to be generated from the root up, with one query per node, and just returning the query for large trees could take a very long time. To mitigate this problem, trees were cached, but there was still a fair amount of server-side computation.

For the new remix trees, the goal was to lighten the load on the servers and push computation to the client side. Before I started the project, the team had decided to maintain a MongoDB NoSQL
database [10] particularly for the remix data. Each object is keyed on the root project id, and the hierarchy is flat, with project data for each project in the tree on the first level rather than in a highly nested structure. This allows adding projects to be fast (append project, grab its parent and append to its child id list), and gathering the right data together to be a simple matter of sending the object as-is. This object is converted to JSON by the Scratch website's Django framework, which is parsed into data by the client.

5.2 Libraries

Because I felt that no existing implementation did quite what I wanted, for better or worse I decided to build Scratch's new remix tree from scratch. The first big question was, what to implement the new remix tree in?

Initially, I actually built a working tree model in Adobe Flash. I personally had several years of experience with Flash, and at that point, the Scratch team had spent the last two years porting Scratch to Flash, so there was plenty of help to be had. With both projects and visualizations in Flash, they would be able to talk to each other. I also liked Flash as an artist's tool, since most alternatives have a very bad if any visual-tweaking interface.

However, many people on the Scratch team agreed that the decision to use Flash made much more sense two years ago than it does now, since Apple and other major companies were moving away from Flash. JavaScript had better cross-device support, and JavaScript, CSS3, and HTML were gradually moving toward a feature set with affordances similar to Flash's. The team was already looking toward the next rewrite of Scratch, and encouraged me to explore options like HTML5 and SVG.

After sifting through graphics libraries, I narrowed down my options to RaphaëlJS [11] (an SVG library) and HTML5 Canvas. Raphaël stood out as an older library with a very good range of browser compatibilities and a fair-sized user base that I could bounce questions off of. On the other hand, everyone thought Canvas was the future, so it was going to catch up quickly. Additionally,
it was considered acceptable for the remix visualization to fail on older browsers, as Scratch 2.0 was also starting to drop support for old browsers.

Raphaël won out due to SVG. I wanted to use vector to have crisp lines while zooming indefinitely. Canvas can handle SVG, but for every update, it needs to redraw the entire canvas, whereas SVG can update just the one DOM element. A friend of mine had implemented a data tracker for a company recently, and told me he was able to render at least twice as many graphs by switching from Canvas to SVG.

For other libraries, I didn’t know where to start, so I picked up an HTML5 starter pack containing jQuery, EmberJS, Handlebars, and Underscore. Later I switched from Ember to Backbone to match the infrastructure of the rest of the website.

5.3 Splitting the model and view

Being a tree, it was natural to want to draw it recursively. Of course, child branches can't be positioned until their parents are positioned, because they have to sprout out of them, so I needed some sort of branch class that drew its children.

Here was the difficulty though: we would like our branches to not cross and create an unusually dense, messy trees. Therefore if a project has more children, its branch needs more space around it to make sure the branches don’t cross: essentially, the triangle bounding a project and its descendants should not overlap the triangles bounding each of its siblings’ projects and descendants. Therefore the parent branches can’t be positioned until the children are positioned! Thus no actual drawing of branches could happen until the entire tree had been traversed so that all branch positions could be calculated. Being used to MVC (Model-View-Controller) front-ends, I found this to be a very natural place to split the model from the view-controller (henceforth simply referred to as the view). Thus the BranchModel takes the raw data and from it builds the structure of the tree, calculating each branch’s position and angle, and finding a bounding box for the entire tree.
5.4 Organization of the canvas

The canvas can’t actually be drawn until the BranchModel finishes, because it needs the bounding box info from the BranchModel to determine its size. (This is the canvas’s “absolute” size. On the web page, the box containing the tree is always of constant size, but the user can pan and zoom around the tree in that box. However, the absolute size of the canvas is the coordinate system in which the SVG elements in the DOM are actually positioned.) To ensure that there won’t be white stripes on the top and bottom or left and right of the tree, or that portions of the tree don’t get cut off when zoomed all the way out, the proportion between the width and height of the canvas need to match that of the box through which the user views the tree, henceforth referred to as the “viewbox”. Thus the bounds from the BranchModel are padded to match these proportions, and the tree position is adjusted so that it will be centered.

Nothing can actually be drawn until this canvas is initiated. At this point, the canvas calls upon the functionality of the Raphaël library to render itself. The view is drawn in layers.

- At the bottom is the background layer, which simply displays SVG paths designed in Adobe Illustrator.
- Above that is the tree layer, in which the branches are rendered.
- Above that is the pan and zoom layer, which catches navigation actions.
- Above that is the pageturner layer. If a node is paginated, the branchlike elements that capture page turns and tell you how many projects are left to the right or left go here.
- Above that is the thumbnails layer, which contains the preview thumbnails, and the glow and hearts behind them. Upon hover, it also contains the info box.
- Above that is the text layer. For reasons explained in Section 5.9, this layer is not actually rendered within the Raphaël canvas, but in a div element that exactly overlaps the Raphaël canvas.
Finally, the layer that contains the zoom buttons is on top.

5.5 The tree layer

The tree layer takes as input the BranchModel, and creates a BranchView, which traverses the model and actually places each element in the locations pre-calculated by the model.

Figure 5-1: A project with more spread-out children needs more angular space to avoid intersecting projects around it.

5.5.1 How do you avoid crossing branches?

Even with the above implementation ensuring there is enough space between siblings, it’s possible for branches to end up overlapping if they simply wrap around. For instance, the leftmost chain
of branches may curve around until it intersects with the tree, or sinks a portion of the tree below the boundary of the canvas, where it becomes inaccessible.

Figure 5-2: A Pythagoras Tree, an example of a tree in which the branches curl around to the extent that the tree self-intersects.

We can avoid this by maintaining the invariant that no branch have any children under the line perpendicular to its top. That is, the spread of all the descendants should be no more than 180 degrees.

I use a greedy algorithm to partition a branch’s children into pages. In BranchModel, if the children ever cover more than 165 degrees altogether, I attach page turners to the branch and start adding children to a new page. Additionally, that branch starts keeping track of what page it’s on, which is the first one by default. After the tree is drawn, I traverse the branches and find the featured project, then change all of its ancestors’ page numbers to allow the featured project to be showing. Then I traverse the tree again and hide all branches and project thumbnails that are not part of their parents’ active page.

5.6 Pan and zoom layer

Pan and zoom are both implemented using Raphaël’s setViewBox function, which allows me to change what portion of the canvas is showing. In the final product, pan is implemented by detecting click-and-drag on an invisible layer between the project boxes and the tree branches. This allows users to click and drag over the branches and background details that don’t take click events.
Figure 5-3: To avoid self-intersection from curling around, the spread of the children of any branch should be no more than 180 degrees.

Zoom is implemented by detecting scroll on the same layer and changing the view box size by a fixed positive or negative percentage (based on direction of scroll), with the point under the mouse constrained to be fixed. The zoom buttons do the same, but assume that the point to zoom toward is whatever is in the exact center of the screen. Users can’t zoom beyond the edges of the canvas, but can zoom in indefinitely.
5.7 Page-turner layer

The Pageturner object handles rendering the page turners, updating the page number of a given node in the model, and handing the animation and appearance of the changed page when clicked. Its arms display the number of children total to that side of the node, that is, the number of children not shown in that direction plus all the children's descendants. In the model, each node keeps an array of these values pre-calculated for every page, and simply looks up the new one and updates the text when the page is changed.

The animation is important usability-wise to understand what is going on when the pageturner is clicked. In appearance, it looks like the current set of children rotates and fades out (rotating to the left if you page right, and right if you page left), while the new set of children rotates and fades in to take its place. This is actually a tricky operation.

5.7.1 Page-turn animation: Trouble with “rotate”

First, Raphaël’s “rotate” animations are actually interpolations. That means that if you ask it to “rotate” a set of branches 180 degrees, rather than actually rotating they will shrink down to their base and then grow out flipped on the other side. To actually create a rotation animation, I wrote a function that discretizes this animation into a configurable number of Raphaël animations. For instance, to appear to turn 180 degrees, you might call turn 30 degrees 6 times. The distortion in branch size still exists, but is less noticeable.

However, it would appear that there is some optimization with the SVG animation that does not work well when the animation is discretized. The single animation is smooth, while the bunch of animations one after another take pause between animations, making it appear jumpy and actually take much longer than the smooth animation to finish, even when the timing is set so that it should take less time.

For this reason, the animation is actually a single interpolation rather than a bunch of them, and the change in branch size is slightly visible. Instead of the branches rotating around 180 degrees,
they actually rotate 30, so that the new and old branches are almost overlapping as the old one fades and the new one appears. In practice, users liked this short rotation animation better. It still gives a strong sense of what’s going on, but looks leisurely rather than having objects zoom around the screen or take a long time to respond to the click event.

CSS3 transforms include rotations and are hardware-accelerated, but do not yet support SVG elements. I’m hopeful that we’ll get there someday.

5.7.2 Page turn animation: Accidental hovering

Throughout the animation, the thumbnails on the branches still have their click and hover events attached to them. We don’t want the hover box to appear or users to accidentally click through to a project while everything is moving, so during the animation a cover is thrown over the canvas to block click events. When branches are duplicated (so that we can show the old and new set of branches at the same time) or new branches are drawn, they are drawn by default over the cover, so the cover must be continually pulled to the front.

5.8 Project thumbnails

The project thumbnails are added on another pass of the tree rather than at the same time that the branches they’re on are drawn, because the rotation portion of the transformation to branches and page turners does not apply to projects. Also, it is surprisingly easy for project thumbnails to accidentally get covered by other things if they aren’t drawn at the end.

5.9 Text layer

Rather than being an SVG element like the layers drawn in the Raphaël canvas, the info box layer is an HTML div positioned exactly over the canvas. This is a work-around for a bug in Raphaël. SVG does have a <text> element, and it takes in a font size. For whatever reason the font size is rounded to the nearest half-integer. Thus as a user zooms into smaller and smaller project previews, the
font size in the info box needs to become smaller and smaller, since it is smaller relative to the absolute canvas size, even though the user is zoomed in and the text is always actually rendered at the same size. After 10 or so generations, the font size would round to 0, and text would no longer show up in info boxes.

Thus, while the info box is drawn in the Raphaël canvas, the actual text is created using an HTML <p> element in the text layer. Unfortunately, much care needs to be taken to make sure that the layers move together. On the bright side, any characters that can be rendered by the browser are now supported by the remix tree.

5.10 Not showing hidden projects

As discussed in section 4.6, we decided to remove all non-necessary hidden projects from the tree. Non-necessary means that the project has no active descendants.

At first glance, it seems like traversing the tree and trimming the hidden leaf projects will do. However, it’s possible to have chains of hidden projects, in which case trimming the hidden leaves may leave behind a hidden parent project that is now a leaf.

This must be dealt with in BranchModel, because trimming branches causes the positions of other
branches to change, and BranchModel needs to calculate those positions. Each BranchModel creates all of its children BranchModels. If a BranchModel’s child has only hidden children, easy; skip creating that child and continue. Otherwise, once the child is done being generated, if it both corresponds to a hidden project, and ended up with child count 0, remove that branch from this BranchModel’s children. In this way, all non-essential hidden projects are removed from the model.

5.11 Efficiency

Some users complained about efficiency of the initial release. The remix tree ran pretty smoothly on Chrome 28, but 2-3 times slower in Firefox 23 (noticeable, but still reasonable-looking and usable), and significantly slower in Internet Explorer 10, to the point that it was impractical to use on medium-large trees. While the slower performance in IE was inevitable, due to IE having only just begun to officially support SVG in version 9, between launch and 8/19/2013, IE users have made up 21% of all visitors, and this figure is expected to increase with the start of school, since school computers may only have IE available. Having the feature perform poorly for such a large fraction of the audience seemed poor and could negatively affect Scratch’s image. (Addendum: After changing the text rendering from Subsection 5.11.2 to the current model discussed in Section 5.9, most IE animations now perform better than Firefox’s, on par with Chrome’s, although the animations for the page turners, which still use SVG text, are still extremely slow, don’t catch hover-off, and require double-clicking in order to register a single click. It seems that my motivation was incorrect; general efficiency in IE’s SVG is actually pretty good. I just have to hope IE fixes its bugs.)

On every browser, even Chrome, extremely large projects caused the browser to stall, either popping up the “Abort Script” dialog box or crashing the browser altogether. Despite the low occurrence of such large remix trees, they are the most popular to view. The remix tree for “Pong 2”, a project with over 14,000 remixes, causing crashes was the most reported bug about the remix tree.
5.11.1 Dealing with huge trees

The tree was originally implemented so that every page was drawn, and then the ones that were not showing were hidden. However, they might eventually be revealed again, so they were still in the DOM. I had imagined that implementation of SVG would skip over these elements in the rendering process, but it appears that the way the SVG viewBox works, it needs to recalculate paths for all elements whenever the view changes, even if those elements are hidden.

The answer was to only actually generate those branches that the user would view. The raw data for our largest tree was less than a megabyte in size. We could compress this, or ask for only chunks of tree data at a time from the server, but this meant more work server-side and a higher rate of requests. The size of the raw data wasn't really the issue. A typical page with Scratch comments contains several megabytes of user icons and preview thumbnails.

The view had to be re-written so that the draw function could be called multiple times, and could draw the sub-tree with root at any arbitrary node, not only the actual root of the remix tree. The way the view was drawn sped up quite a bit; rather than the entire model, the view simply drew whatever was on the active page. Changing pages actually deleted the elements of the previous page and rendered the new one. The load time moved from all at the beginning to small chunks during page-turning. With this change, besides the extra couple of seconds due to the download time of the raw data, large trees behave almost as well as small trees. Typical time after clicking before page turning begins is a few hundred milliseconds (where it was 0 before).

5.11.2 Text

The initial fix for the zero font size problem discussed in Section 5.9 was to convert the font into SVG using the service Cufón. [12] Firstly, this forced users to download the font file, which was about 4MB to have an extended alphabet (still not nearly enough; Scratch is available in over 50 languages) and still 1.5MB for only the characters in Latin-1. Then, it took time to trace out each letter when hovering over project thumbnails, causing the info box to go from appearing
instantaneously to not appearing until half a second had passed (depending on the length of the
title—long titles were clipped to a rectangle that fit in the info box, but Raphaël still needed to trace
out the entire title underneath). One experiment was to front-load this, pre-rendering all the titles
on the page at the time that the page turner is clicked. This was somewhat disastrous, causing
page turning to potentially take minutes. In the end, abandoning this route altogether was the
way to go.

5.11.3 Reloading thumbnails

It used to be that preview thumbnails were splayed out overlapping each other. I didn’t want
the info box pop-up to disturb the ordering of the elements, which would happen if I created the
thumbnail in the info box by pulling the existing preview to the front and resizing it. So I would
duplicate that element to create the thumbnail in the info box, and destroy it on hover off.

This worked fine in other browsers, but in Chrome, upon hovering over the thumbnail, the image
would disappear for a second before reappearing, sometimes not reappear, and sometimes reappear
as a broken image. It turned out that every time the duplicate was created, Chrome was asking the
server if the image had been updated since the last download rather than just trusting the cache,
even though the cached images had not yet timed out.

This was fixed by my switching back to the first plan. In the current design, project thumbnails
barely overlap, and it doesn’t matter much if they change ordering.

5.12 Logging

To collect data for Chapter 6, I needed to implement logging. I was considering using Google ana-
lytics events, but those only provide cumulative results and in fact the terms of service disallow
collecting identifying personal information. We wanted to leave open of aggregating personal data
open (see Section 8.2.3), so I decided to save data to Scratch’s own databases.
This is done by ajax request at page load or page unload. Scratch had an existing style for log items, called "actions", which has fields that preserve the idea of the action as a sort of sentence.

```json
action = {
    'tags':
    'actor_username':
    'actor_image_url':
    'verb':
    'object_type':
    'object_name':
    'object_id':
    'object_url':
    'target_type':
    'target_name':
    'target_id':
    'target_url':
    'ipaddress':
}
```

The only interactions the user has with the remix tree are navigation to and from project pages, and navigation within the remix tree, such as hovering and zooming. The existing fields were sufficient to describe these actions, and so I used the same format, and saved the actions to another MongoDB.
This section provides an analysis of some interesting data gathered during the few short months of the remix tree’s life so far. I hope to glean some insight into the design’s strengths and shortcomings that just user feedback doesn’t necessarily provide.

I must begin with a disclaimer. These results are taken from a production website. There is very little control over variables, such as a sudden influx of users of a certain type from a conference. There’s no good way of setting up a control, as I can’t impede other developers who are constantly rolling out their own features all around the site. And even then, Scratch is a learning tool, and our team is concerned that running multiple versions of an interface will bewilder teachers or be unfair to students.
This is especially true for section 6.2, in which we compare statistics of the new remix tree, embedded in the new Scratch 2.0 site, to the previous remix tree. While it’s natural to begin by asking how the new remix tree stands up to the old one, these are visualizations written with vastly different goals in mind, and surrounded by even more disparate environments.

### 6.1 Timeline

![Timeline of updates to the remix tree.](image)

**Figure 6-1:** *Timeline of updates to the remix tree.*

This timeline shows major updates to the new remix tree. Since many data here are gathered with respect to time, keeping this timeline in mind when viewing those graphs may help us figure out what spurred a change in trend. (Once again, keeping in mind that many other updates occur on the same days.) The previous remix tree was launched in 2010, but we will only be using much more recent data.

Scratch 2.0, which had been available in beta since January, became the official Scratch website on May 9, 2013. A partially working remix tree had been available for the last two weeks of Beta, but the new remix tree wasn’t officially released until a week after the launch of Scratch 2.0. Before then, the remix tree page had read “Coming soon”.
6.2 Comparison to previous remix tree

The total number of pageviews has actually gone down by 35% from the previous remix tree, despite the fact that pageviews across the entire site have skyrocketed (as expected due to bringing the editor online). This is a steep drop and somewhat surprising given the lack of negative feedback on the forums (although those tend to be more experienced users). It may be that the new trees are more difficult to find, they are hard to use due to bugs and efficiency issues, that internationalization has not been implemented yet (the tree always displays in English and for a long time didn’t display foreign characters, though only 76% of visitors’ browsers are set to English), that part of their niche is covered by some other features, or simply that we’re near launch and users are still acclimating to new features.

The disproportionately large population of Chrome users suggests efficiency issues may be the cause, since the remix tree was primarily developed on and runs fastest on Chrome. Then again, remix tree users and Chrome users may be correlated for different reasons.
6.3 Comparing to other remix displays

There are now several avenues through which users can find remixes. Which ones are they using?

We assume each user has a uniquely identifying IP address. This graph looks at each IP address that was active on Scratch during the logging period, and asks if it has ever used a given method of viewing remixes. That is, if one user uses the remix tree once, and another uses it 50 times, both are weighted the same here.
61% of users stick to one method. Almost a half find remixes exclusively through the remix bar which appears on remixed projects’ pages. Only about 12% have ever used more than one method, and only 2% have tried all three.

### 6.4 Returning users

I took some statistics on the remix tree’s retention rate. Unfortunately, it is all too similar to our community graphs for the number of comments or projects users post.

A massive number of users view the remix tree once or twice, and then never return, while a small core audience has seen tens to hundreds of remix trees. Surprisingly though, there is a long tail of dedicated visitors who have looked at an average of nearly 80 remix trees per day.
Figure 6-5: The frequency with which users of the remix tree have continued to use the remix tree. For instance, 13,163 IPs have visited the remix tree once and never returned, while a dedicated few have viewed thousands of remix trees over the last two months. Note that both axes are logarithmic in scale.

6.5 Navigating the remix tree

To look at how users used the remix tree, and to watch out for usability problems, I logged the various actions users can take on the remix tree. About 50% of remix tree visitors click through to visit one of the projects displayed in the tree, and the other half hit the back button or otherwise navigate away.

These figures are graphed with respect to date to try to see if updates to the remix tree caused any changes in user behavior. For instance, one might ask, did adding love-it indicators cause users to become more curious about certain projects, perhaps making them zoom or hover more often?

Most of the data looks about consistent. However, happily there is a huge spike in the number of projects hovered over corresponding to July 19, when we increased the size of project boxes and made them easier to hover off of. Average projects examined jumped from 5 to 7.7.
(a) For each session, the number of times a user has moused over an info box.

(b) The number of times a user has moused out of an info box.

(c) The number of times the user has clicked and dragged the tree to pan.

(d) The number of times the user has continuously moved the scroll wheel to change the zoom level of the remix tree.

(e) The number of times a user has pressed the zoom in button at the top left of the remix tree.

(f) The number of times a user has pressed the zoom out button at the top left of the remix tree.
The number of times a user has pressed the zoom equals button at the top left of the remix tree, causing the tree to become zoomed all the way out. Note the change in scale for the frequency axis on this graph compared to the other graphs.

The average number of times a given action was taken during a single session on the remix tree.

The number of sessions in which the user used a particular method of scrolling, and how often they used both or neither.

6.5.1 The average session

The average session was on a fairly large tree, because the user hovered over about 6 projects before clicking on something. Yet in the average session, a user didn’t bother to use the pan or zoom tools at their disposal. Perhaps once if they had to to see more of the tree, but the impression I get is that it’s too much of a hassle.

My worries about the discoverability of scroll to zoom might have some backing. In my personal experience, zooming via scroll is much handier and quicker than zooming with the buttons, yet the buttons get more use. On the other hand, a user can zoom to just the right level using a scroll wheel, while they would have to repeatedly pan and zoom if exclusively using buttons, so the buttons may be over-represented.
Finally, Zoom Equal is arguably useless. Personally, I like it when getting my heading on very large trees, but it can also be replaced by scrolling. The constant low frequency might mean that it's the sort of thing you'd only need to use once a session on very large trees, or it might mean the button is being hit by accident, which would be very annoying if you were looking at something and now have to zoom into it again. This can be resolved with more user feedback in the future.
This chapter assesses the success of the project in terms of the goals outlined in Chapter 2. I've split each goal into Good, Medium, and Bad points based on my conception of what parts need the most improvement.

### 7.1 Basic functions

**Good:**

- Be able to easily navigate to a remix tree that might be interesting, and from there easily navigate to a project that might be interesting. Links are a click away and where you would
expect them.

Medium:

- Be able to see projects in the context of remixing. The relationships between projects is clearly visible. However when there are too many siblings and large parts of the tree are hidden, it can become difficult.

- Be able to easily browse projects. Getting to and from the remix tree is easy. Finding older projects and more popular projects is easy because of the sorting and visual indicators on the tree. However, this may cause a sort of tunnel vision directed at those focused projects, and less popular projects have much less visibility.

- Be able to see useful details about projects. The thumbnails and love-it counts are easily visible at a glance, and the title, author, image, favorite and love-it counts of projects are accessible by hovering. Users may still want to see project descriptions or hints as to what changed.

- Be able to see differences between projects. This is apparent for mostly-visual projects, but may be challenging to see for projects with small visual changes, and may not be indicated at all for code-heavy changes.

7.2 Usability

Good:

- Safety

Security isn’t a huge concern, as the remix tree is read-only. In terms of undo, hover on/off are easily reversible, as are clicking on links by accident.

Medium:
• Efficiency

Huge trees render relatively quickly and animations are snappy across all browsers. Performance in Internet Explorer is much slower than in Chrome, but still has reasonable timing, and Internet Explorer is expected to continue to expand its support and efficiency with rendering SVG graphics.

Efficiency of navigating the tree does suffer somewhat for large trees, as paging through branches and dragging the tree around when there are long chains takes time. Hopefully the most relevant browsing results are presented to the user first (the tree prioritizes showing the neighborhood of the project from which the tree was navigated to, older projects (which therefore have longer histories and often longer chains), and popular projects). Without showing the projects in an entirely different format, showing what’s most relevant is the best we can do for large trees.

Efficiency and compatibility suffer on some browsers, especially mobile.

• Learnability

All clickable things show a hand cursor, and a move cursor and zoom buttons invite the user to start exploring the visualization. The interface is very simple, with no instructions, hidden settings, shortcuts, or options panel. We’ve also sought to show only the most important information, ditching for example project descriptions and stars to declutter the view and keep it from looking overwhelming. The navigation interface copies Google maps, which is probably familiar to users. It therefore has the mouse scroll mapped to zoom (about the point the mouse is on). However, this isn’t indicated on the remix tree page.

I would think I was doing well in terms of learnability, except that the significant drop in the remix tree page discussed in Chapter 6 leads me to suspect something is getting in the way. Perhaps the colorful tree-scape is too abstract after all? I wish I had more user feedback on this point.
7.3 Serve user goals

Good:

- The host of an art competition might want to glance art changes over time. All entries are now clearly visible.

Medium:

- A browser might just want to see something cool, or might want to find inspiration for their own project. He or she can see popular projects at a glance from the number of hearts around them. However, popular projects may be buried in the hidden parts of the tree. This person would want to sort the tree by a popularity metric rather than age.

- A remixer seeing a "Remix this!" project might be looking for the correct project to remix; possibly the latest, the best-written, or the one most similar to what they have in mind. Most likely, the longest chain is attached to one of the first remixes to be shared, and therefore is easy to find from the remix tree. Finding a popular project has advantages and disadvantages akin to the browser’s in the item above. For similar projects, finding one with the right look should be easy, but one with the right functionality and lack of bugginess is difficult.

- An analytical thinker might just want to look for patterns in how remix trees grow. This visualization is good for small trees, but seeing the overall structure of a large one is tricky. This user probably wants the tree to be more densely packed, preferring to see more of it over having each project be visible.

Bad:

- The writer of a game might want to glance at code/feature changes over time and incorporate them back into his game. He might be able to see changes from the thumbnail, but probably would rather see the code diff and the project description.
7.4 Encourage remixing

![Monthly Project Shares](image)

**Figure 7-1**: The orange line is number of remixes created by month, and the blue line is number of root projects created by month.

Medium:

The number of remixes slumped during the week after launch before the remix tree was out, and quickly flew up once it was back. Hopefully, the return of the remix tree had something to do with this growth. However, the number of root projects being created grew even faster.

The number of remixes is a solid fourth of all projects created. Before Scratch 2.0 it's about the same. In fact, this same time last year happens to be in the middle of a big dip in project shares, at which point remixes made up almost a third of all projects created. So remixing is falling off in percentage, but gaining in numbers.
CHAPTER 7. ASSESSMENT

7.5 Reflection

Nearly all of the problems with the tree come from larger trees where much of the tree's structure is hidden. Some of the problems can be overcome by allowing the user to adjust the parameters of the tree, for example sorting based on popularity or making the whole tree more or less dense. However, the problem is innate to trees, and as remixes are fundamentally trees, it's most natural to represent them with a tree visualization.
The Scratch remix tree is still very much a work in progress. Many of these features have been planned since the beginning, and others have naturally come into the plan via user feedback and data analysis. I intend to continue work on these features, and they are listed under short-term goals, in order of priority. Others are great ideas that manifested during a meeting or on a group retreat back in the initial design phase. Perhaps they were too grand for us to get to, or the project just wasn’t mature enough yet, but they are worth noting. These are listed under long-term visions.
CHAPTER 8. FUTURE WORK

8.1 Short-term goals

8.1.1 Multiple views

Several user suggestions contradict each other because of the space trade-off between wanting to view each individual project and wanting to see overall structure of the entire tree. The idea is to add an expandable options panel to one side of the remix tree that would allow the user to switch between at least two "modes". The first would prioritize project thumbnails. This would be the current view, where the angle between branches is relatively large, so that at most 8 immediate children fit around any node, and any more overflow onto the next page. The second would be dense, potentially doing away with project thumbnails altogether and replacing them with small circles. The user would continue to be able to hover over these circles to see project details, but these could be packed much more closely, perhaps showing 20 immediate children per "page" per node. It would be nice to have a smooth animation between these two modes, but the implementation would be quite complex, and so the whole tree would probably be redrawn when the mode is changed.

A final, very different mode that could potentially be useful would be a GitHub-like visualization, as touched upon in Section 4.1. This would allow users to see detailed timing information about when branchings happened. However, this idea is very fundamentally different from the current one and would require an entirely new view and model to be implemented.

8.1.2 Branch colors

Since early on, changing branch length or color have been suggested as indicators for differentness between projects or newness. Changing branch length is troublesome because it could cause the tree to self-intersect, and also differentness is hard to gauge and easy to game. However, users often need to find the latest project quickly in order to make remixes with long chains, and it might be cool to be able to get a sense of how old certain parts of the tree are, so I was thinking of having branches indicate oldness.
The branches would be colored somewhere between a dark brown and a bright green, with the metaphor being that new branches on a tree start out green. At any time, the root branch (the trunk) would be the darkest color, the newest project would be the lightest color, and colors for other branches would be linearly interpolated based on the date they were shared.

8.1.3 Internationalization

The remix tree has yet to be internationalized. Currently, it will appear in English no matter which language the user is set to. Not only do phrases need to be translated, but we need to give special consideration to languages reading right to left. Currently, the remix tree prefers left-to-right readers by keeping paginated nodes by default scrolled to the left.

8.1.4 Navigation

As discussed in Chapter 4.4, navigation was originally designed so that users could zoom into small nodes automatically, saving them the tedious manual dragging and zooming. This may be worth bringing back. For instance, we could have it so that the first click on the info box zooms the user into a project, and the second sends them to the project, if we are willing to make click-through to projects twice as tedious. Or, we could capture right-click, double-click, or some other shortcut which acts as an instant zoom. Since this wouldn’t be immediately obvious, an instruction would need to be put somewhere near the visualization. In the interest of making it simple and easy to use, we need to take care that such an instruction list never becomes more than a couple of lines long.

8.1.5 Copy sharing same remix tree

In Scratch 2.0, we removed the ability to remix one of your own projects. However, users can create copies of their projects, edit, and share those, essentially performing the same function.

Currently, this copy becomes the root of its own new tree. This doesn’t make sense, since the history and dependencies are still the same.
CHAPTER 8. FUTURE WORK

In the old Scratch, such a copy would become the child of the project that was copied, but in the new Scratch, the original project may not yet have been shared when the copy is shared. Design-wise, it’s odd in this case that the project that is shared first is a child of the project that is shared later, and implementation-wise, the database has no entry for the parent at the time the child is shared, and so the tree would become disconnected. For these reasons we decided that copies should become siblings of their duals on the remix tree, inheriting the original project’s parent as their own.

8.1.6 Tree previews

![Figure 8-1: The current button leading to the remix tree page. The tooltip appears on hover.](image)

A generic tree-like image marks the button on the project page leading to the remix tree visualization. There was some discussion on replacing this with a very small version of the actual tree, or having the actual tree structure appear over the button when hovering. This would make it
Chapter 8. Future work

a little easier to evaluate whether the tree seems interesting before committing to click through. However, it could be tricky for large trees and might be confusing for users.

8.1.7 Toggle hearts and stars

My advisor was worried that a profusion of hearts and stars in the background would make it difficult to see all the project thumbnails at a glance. The original design for this feature was to be able to toggle the visibility of hearts and stars separately, with hearts on by default but not stars. So far, these background elements haven’t caused any trouble, so this has been put off.

8.1.8 Toggle hidden projects

Note that in Section 4.6 we ended up removing leaf unshared project nodes, since the majority seemed to think they shouldn’t be shown. However, they had been shown initially because a Scratch team member thought they might be interesting to see, and there is a user feedback request to be able to view them. If we add multiple views to the remix tree, another item on that options panel could easily be re-rendering the tree with all the hidden projects shown, since all the data exists there.

There is some trouble with this; for a period, there was a bug that caused every project to be shared as soon as it was created, causing massive numbers of grayed projects to accumulate on trees. The bug was fixed, but the data is still there, and they are indistinguishable from projects that were in fact shared and then unshared. Thus projects existing during the time period of this bug would show an inaccurate accumulation of unshared projects.

8.2 Long-term visions

8.2.1 Growing trees

The time slider was one of the most inspiring ideas behind having a nature-based tree in the first place, and it has yet to be implemented. The start of the slider would be labeled with the date of
creation of the root project, and end at today. Small nodes along the slider would indicate when projects were added to the tree. The slider would always be all the way to the right (current time) by default.

This growth could be done by completely redrawing the tree when the slider is moved. However, long loading times between changes would make this feature tedious, and the user’s position in the tree would be lost every time they changed the time. I believe it is important that the growing tree be a smooth animation for the user to get a clear idea of how it is growing and not find the interface too frustrating to use.

Moving the slider would cause the tree to grow or shrink. For instance, moving it to start would make it become a tree with a single node. Moving it past a small node would cause that branch to animate, growing longer and having the project thumbnail pop out of it. Moving it past a bunch of nodes would play branch growth animations until the tree caught up to how it was at that time. The animations for siblings could play at the same time, but animation for children would have to wait until their parent animation finished. But to keep it from taking too long, the timing of the whole thing would be scaled based on how many levels of children need to grow. Moving the slider backward would play the animation in reverse, with the tree shrinking down. Additionally, the background would indicate the passage of time, with the earliest time being rendered somewhat like dawn, and the latest somewhat like dusk. (This is why the current rendering of the background looks a bit like sunset; it is the latest time in the day.) The clouds would drift right as the day got later, for instance zooming to the left if a user jumped to the beginning.

How would this interface with paginated nodes? One can imagine a project where all remixes are immediate children of the root. Ideally, the first branch would grow out straight, then the second would push the first one a bit to the right to make room, and so on until there are enough branches to cause a page-turner to appear. Then the number on the page turner would start ticking up. But what if you were on page 2 and went back in time to when page 2 didn’t exist?

My proposed solution for this would be to show all animations in-place. That is, have the page-turner showing by default. When the first project grows out of the root, it is drawn where it
finally ends up, angling far to the left. The number of pages need not fluctuate and the rotation of branches does not need to be constantly recalculated. Perhaps the outlines of the final tree could be shown at all times, so that the tree seems to be growing to fit that glove.

Another question is, what would happen to the branch colors, as explained in Section 8.1.2? Instead of staying constant, I think it’d make sense for the colors to get continually updated so that the project under the slider is rendered bright green. As time moves forward, new branches would sprout green and gradually darken. This might be sacrificed if efficiency is a problem.

8.2.2 Forests

![Forest Image](image)

**Figure 8-2:** The forest from *This Exquisite Forest*, encouraging users to explore other trees.

As in Exquisite Forest, [7] trees can be portals to exploration of other trees by rendering some other trees horizontally along the ground or transparently in the background as Exquisite Forest does. It would be best if these trees are somehow related, as in a suggestion engine, and in fact there are two sets of trees that are very relevant to any given project. Scratch 2.0 introduced the backpack feature, which adds dependency edges in a manner that is not tree-like. Every project may be a donor or receiver for code, artwork, or music. It has been suggested during brainstorming that the
donor trees to the focused project show up on the left of the tree, and the receiver trees show up on the right, or that donors and receivers for all of the projects in the tree just show up somehow in the background, or that donor projects show up around the tree somehow, and hovering over them reveals another tree.

Figure 8-3: In this example from Chapter 4, where donor projects appear near the tree, attached with special new edges indicating what was donated. This is one possible implementation of many.
8.2.3 User trajectories

In her thesis on the original Scratch Stats, Chen mentions that one of the major chunks of future work to be done is to add new visualizations, "especially in the personal statistics category." [5] She doesn’t go into why, but in other parts of the thesis states that in user feedback, people got more out of the visualizations when they connect with them personally; one user said, “I can finally see how I grow, Scratchwise.” The original Scratch stats allowed users to view some of the visualizations in terms of just themselves; for instance, they could see a histogram of which blocks they use most frequently. Scratch team members expressed an interest in seeing how these data changed over time as well as in aggregate. There was also concern that there were in general too many visualizations on the ScratchStats page, perhaps discouraging users, and that they should be amalgamated into 3 or 4 major ones.

There was a grand plan to gather personal statistics more or less into one big visualization called the “user trajectory”. This would give each user a general sense of their activity and popularity flourishing over time. It’s also a very interesting subject for the Scratch team, which researches how kids explore and learn.

The remix tree provided inspiration for this second visualization to appeal to nature as well. A user would have a scrolling timeline of their activity, and creating a new project was planting a tree, which grows as it is remixed. Positive contributions to the community, like comments and love-its, could be like planting smaller things like grass and flowers, and incoming love-its and comments could also manifest, perhaps by growing on the trees. On the other hand, site moderators were interested in seeing this kind of visualization to see patterns in flagged (inappropriate) behavior, for instance to figure out if a user should be banned, or notice what kind of pattern accounts that tend to be banned take (as they often create new accounts after being banned.) Of course, this data should not be public. Chen’s thesis mentions feedback with concerns about privacy, and it’s important that users be able to control what in their trajectories is public.

The natural metaphor for user trajectories, as with the remix tree, is meant to inspire users to
CHAPTER 8. FUTURE WORK

build new projects and new connections because they are growing something that they can see the results of. However, for a potentially complicated visualization, it would also be wise to test alongside a more standard trajectory not tied to a metaphor.

8.2.4 Seasonal variation

A fun idea that came out of user testing within the Lifelong Kindergarten group was to have the tree branches decorated with lights during the holiday season. Others suggested having the tree go through real-time changing seasons, and having some Easter egg hidden in the trees every so often, in the spirit of Google doodles. Perhaps the tree could pull elements from the projects it’s base on,
or Scratch users themselves could decorate their trees. This kind of fun change is probably still far
off in the future for our tree, which isn’t very front-facing like the Google homepage (making it a
strange place to hide Easter eggs), and which still needs lots of work on useable features.
Bibliography


