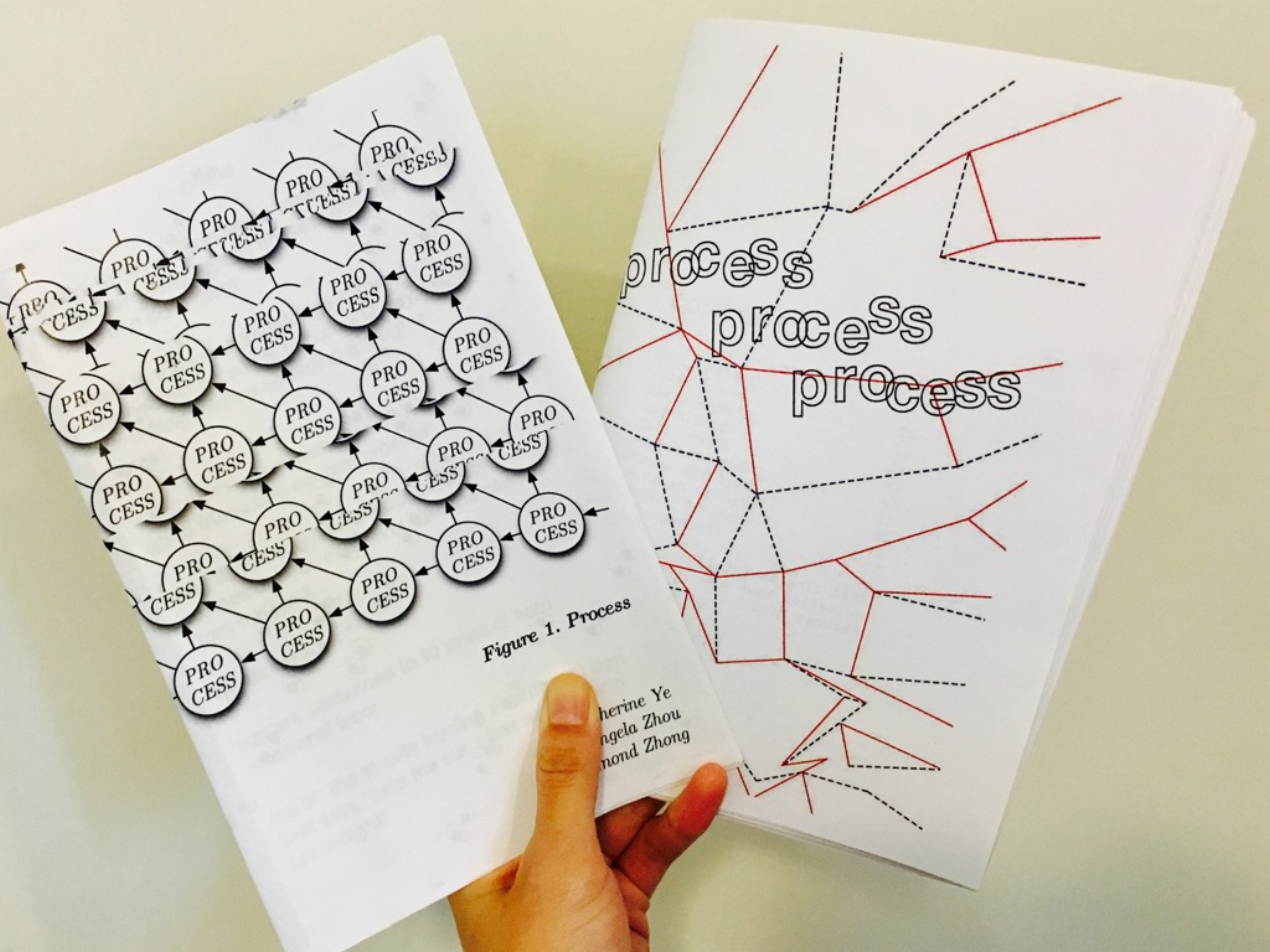
PROCESS

desire paths in creative interfaces

Katherine Ye, with Angela Zhou and Raymond Zhong

*with collaborators Chris Beiser, Robert Ochshorn,
Lea Albaugh, Neeta Patel, Diana Liao, Max Kreminski,
Avneesh Sarwate, Pavel Shibayev, Julia Evans,
and meta-process*


PRO CESS
Figure 1. Process
process
process
process

zine (noun)
a bundle of weird stuff

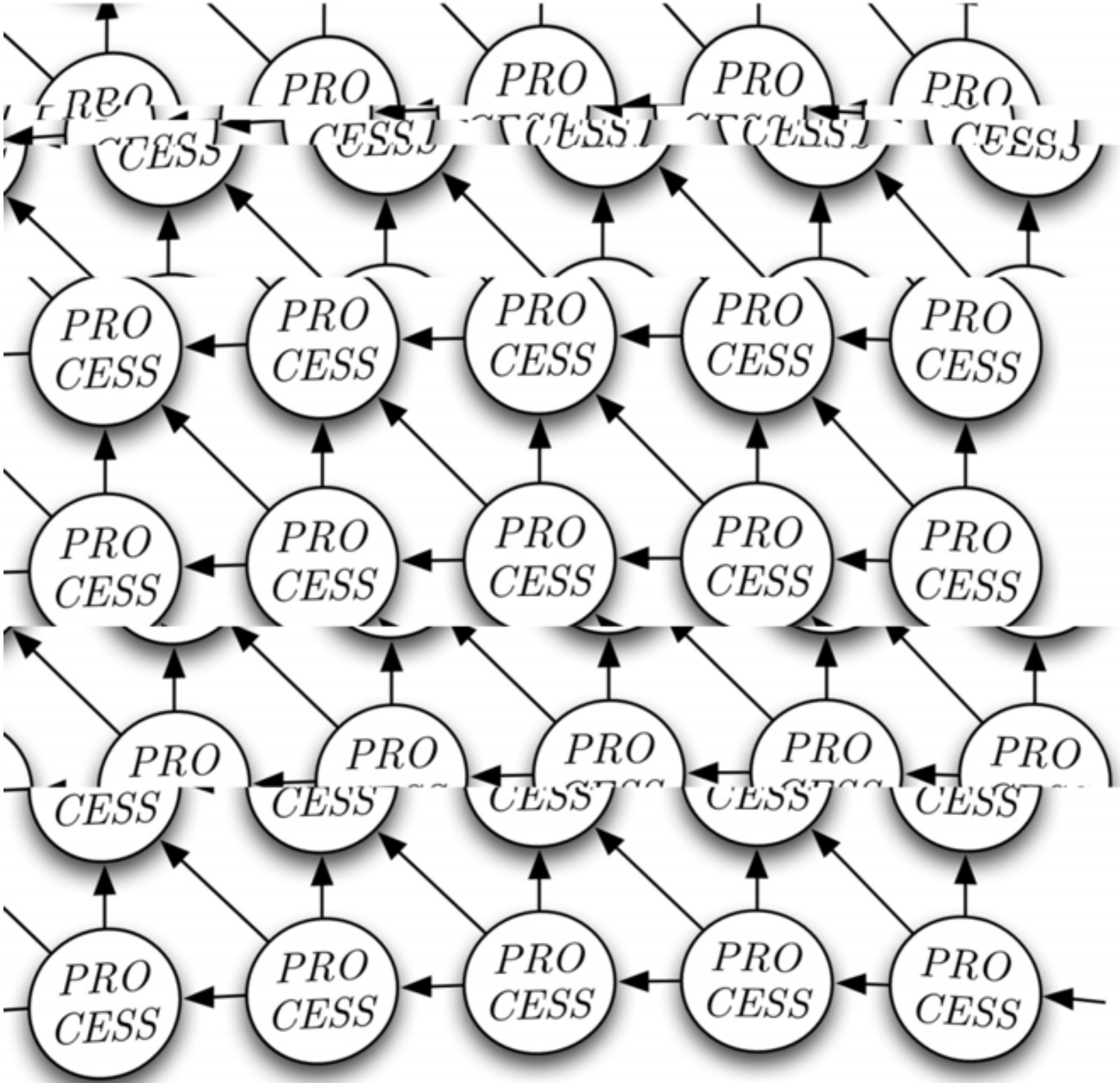
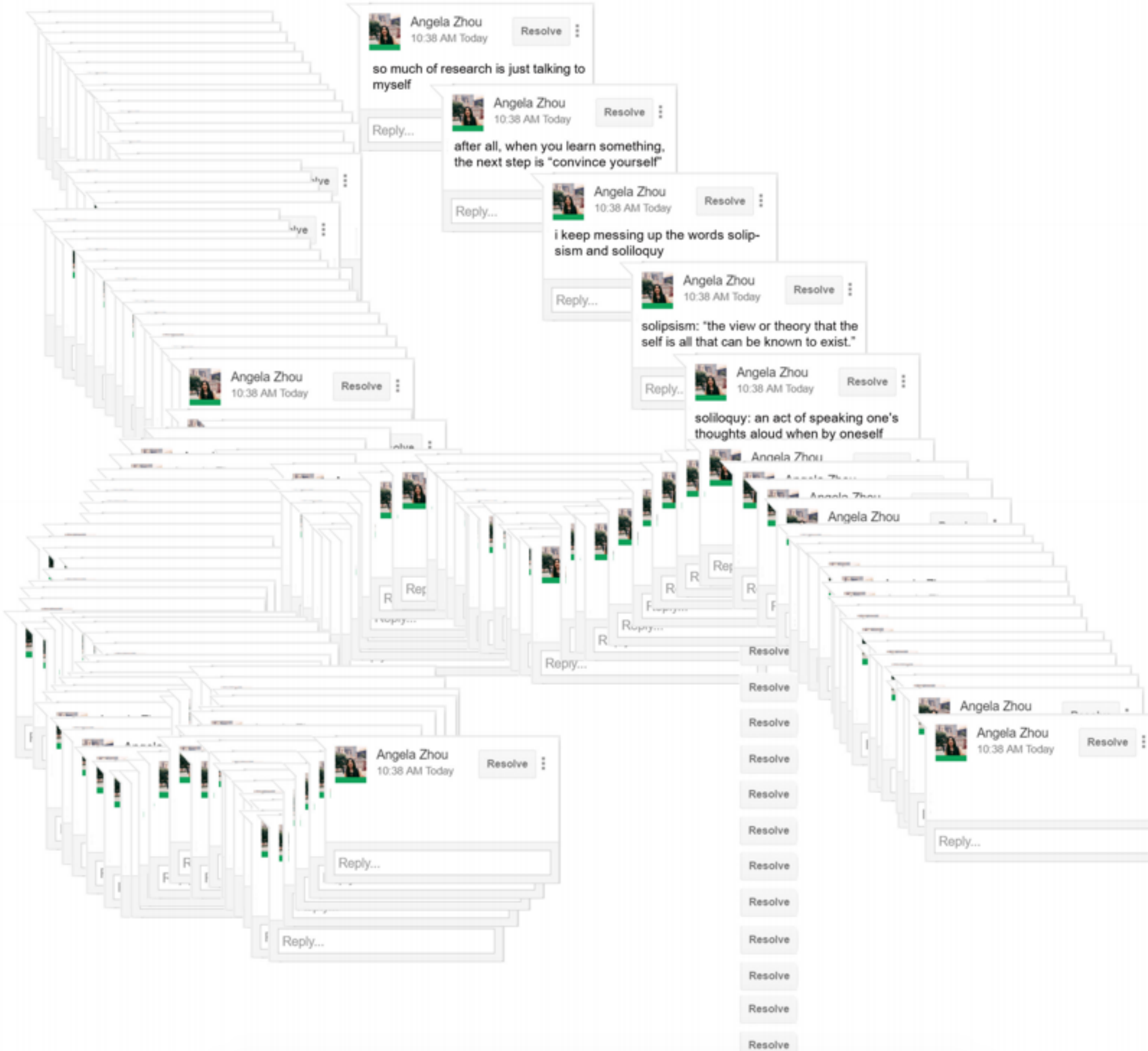
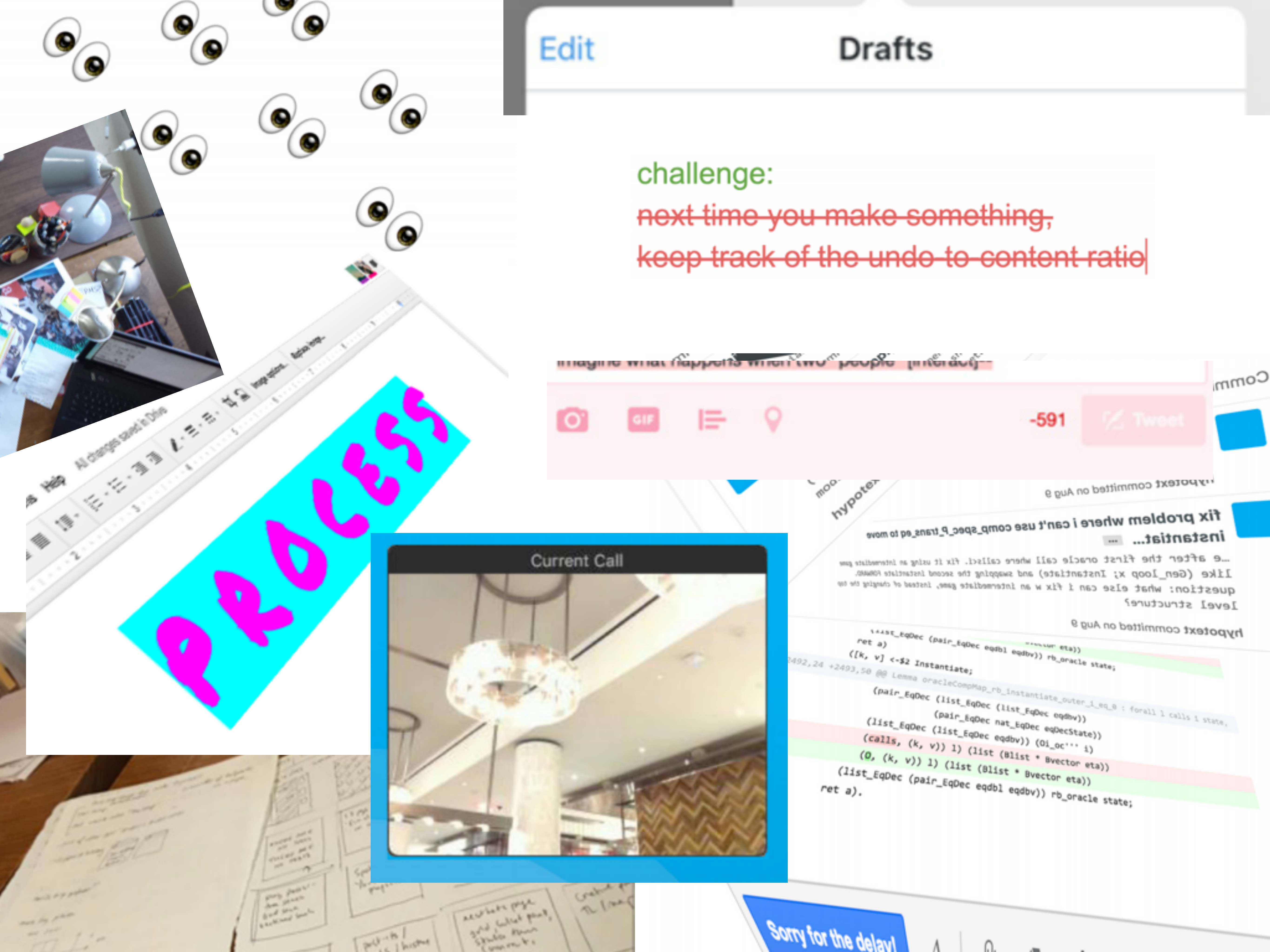


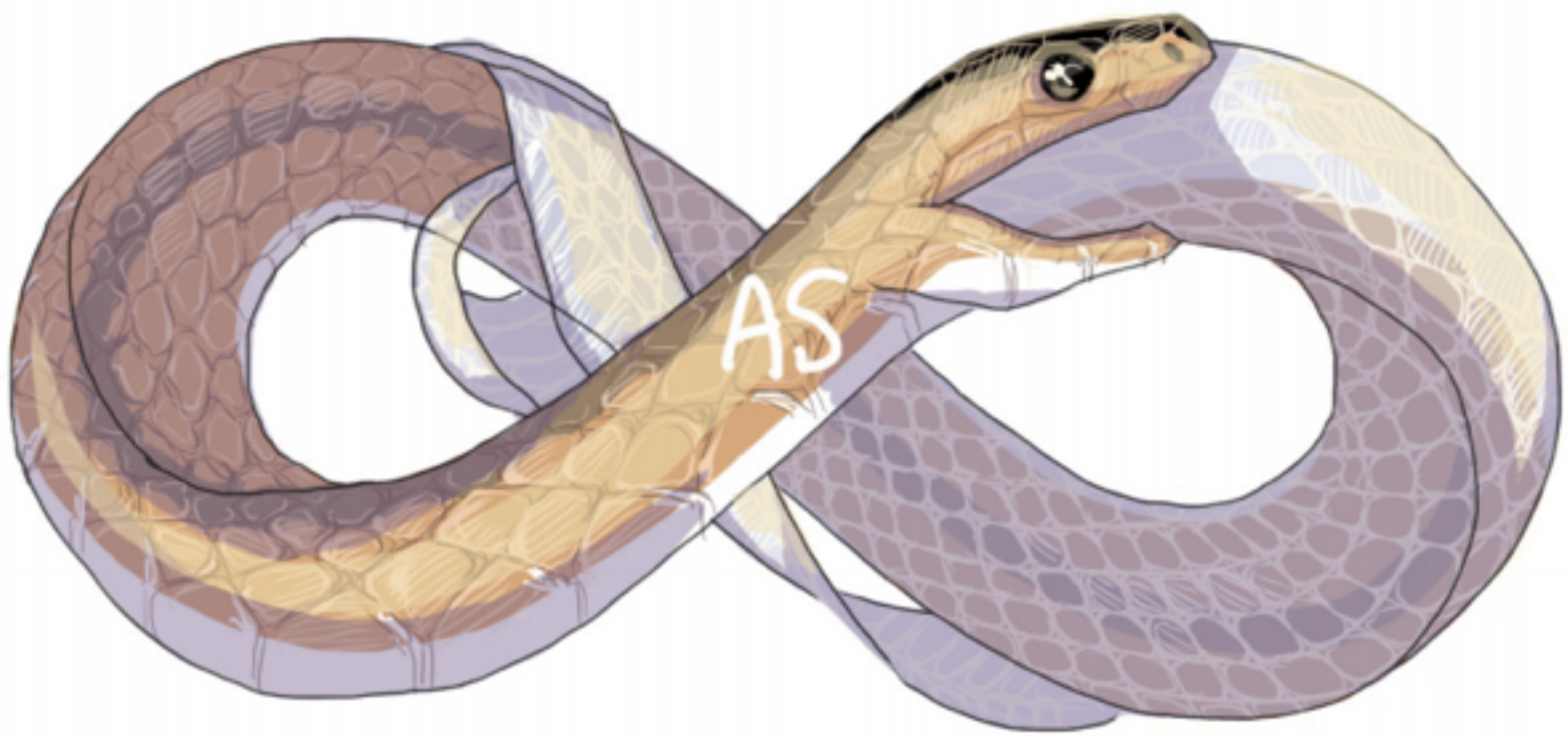
Figure 1. Process


Angela Zhou
10:38 AM Today
Resolve
so much of research is just talking to myself
Reply...
Angela Zhou
10:38 AM Today
Resolve
after all, when you learn something, the next step is "convince yourself"
Reply...
Angela Zhou
10:38 AM Today
Resolve
i keep messing up the words solip-sism and soliloquy
Reply...
Angela Zhou
10:38 AM Today
Resolve
solipsism: "the view or theory that the self is all that can be known to exist."
Reply...
Angela Zhou
10:38 AM Today
Resolve
soliloquy: an act of speaking one's thoughts aloud when by oneself
Angela Zhou
10:38 AM Today
Resolve
Angela Zhou
Angela Zhou
10:38 AM Today
Resolve
Reply...
Angela Zhou
10:38 AM Today
Resolve
Reply...
Reply...
Resolve
Resolve
Resolve
Resolve
Resolve
Resolve
Resolve
Resolve
Resolve
Resolve
Resolve
Resolve

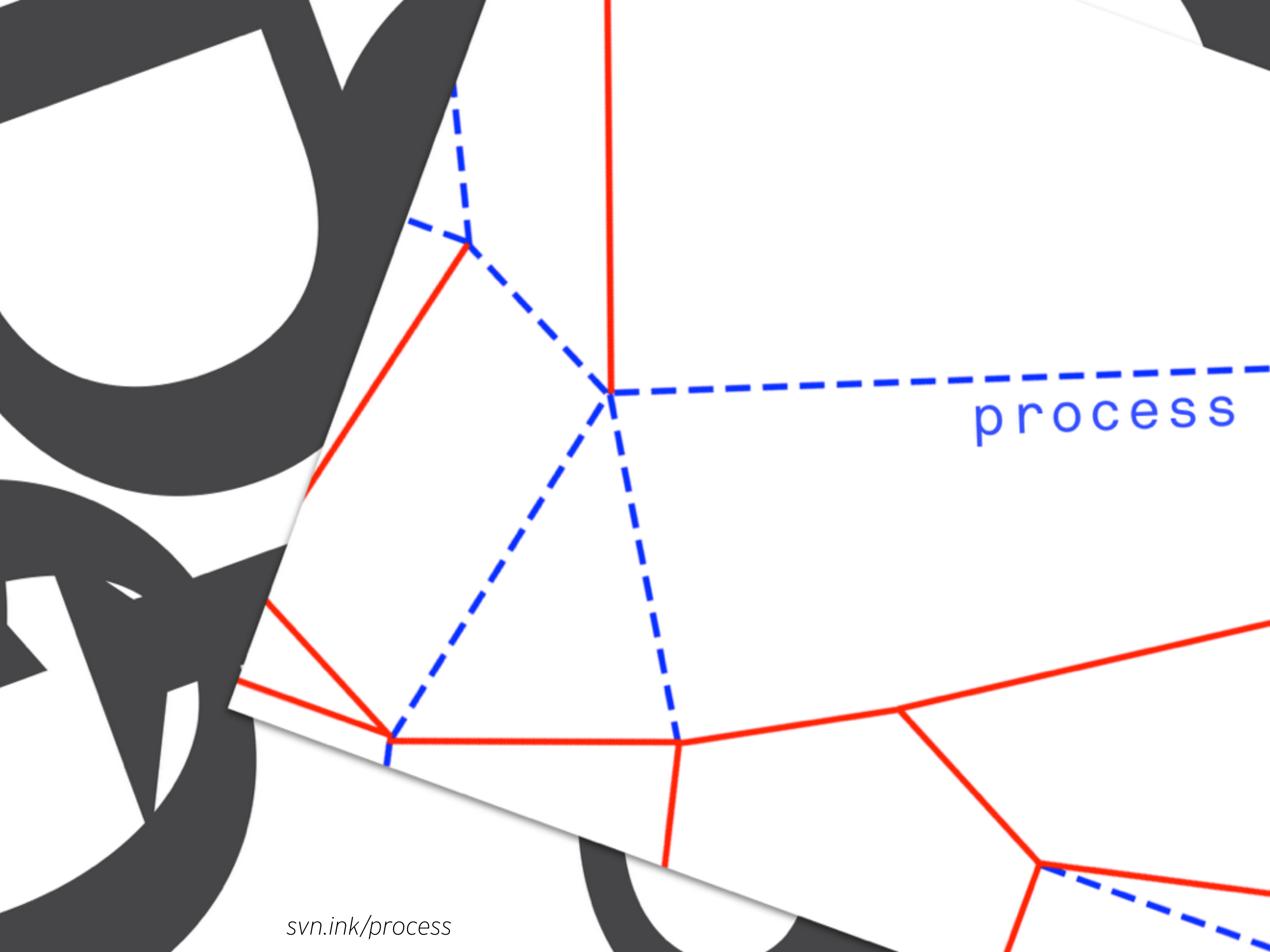


Edit
Drafts
challenge:
~~next time you make something,~~
~~keep track of the undo-to-content ratio~~
All changes saved in Drive
PROCESS
GIF
-591
Tweet
Current Call
Sorry for the delay!

BYPRODUCT



PRODUCT



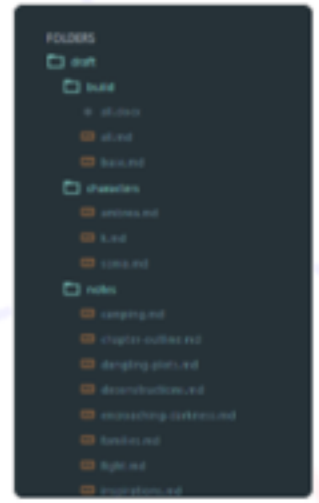
svn.ink/process

Avneesh Sarwate

Avneesh Sarwate is a music technologist, software engineer, and musician specializing in musical interaction design. He is an alumnus of the Princeton Laptop Orchestra.

Chris Beiser

Chris Beiser is a student, some-time interaction designer and software developer. He's interested in versions of the future that are just and meaningful.



Diana Liao

Diana is a college student studying computer science. They make weird narrative video games and pretty art things. And sometimes novels, apparently.

Julia Evans

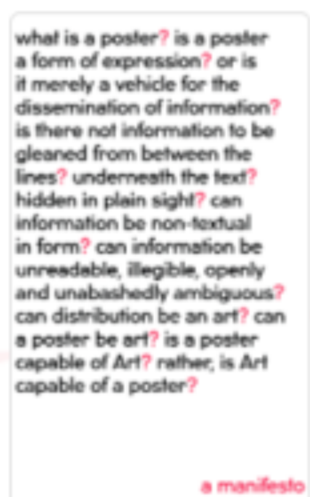
julia evans is a software engineer. she makes delightful & informative zines about programming. you can read them online at https://jvns.ca/zines

Lea Albaugh

Lea builds things in a wide variety of digital and physical media. She is an incoming student at the CMU Human Computer Interaction institute and previously worked at Disney Research Pittsburgh.

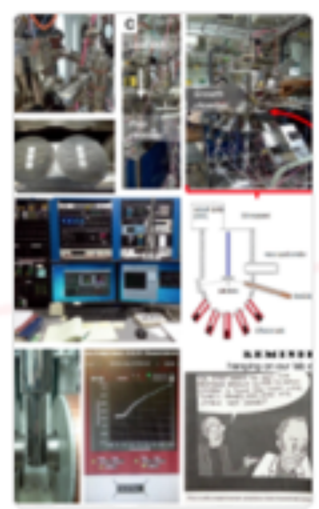
Max Kreminski

max kreminski teaches computers to make things! they are currently a senior at the university of southern california, majoring in game design. this three-sentence bio took them 29 days to write.



Neeta Patel

Neeta Patel is the 2016–17 Graphic Design Fellow at the Frank Lloyd Wright Foundation based at Taliesin West in Scottsdale, Arizona.



Pavel Shibayev

Pavel Shibayev is a second-year Ph.D. student at the Department of Physics and Astronomy at Rutgers University.

Robert Ochshorn

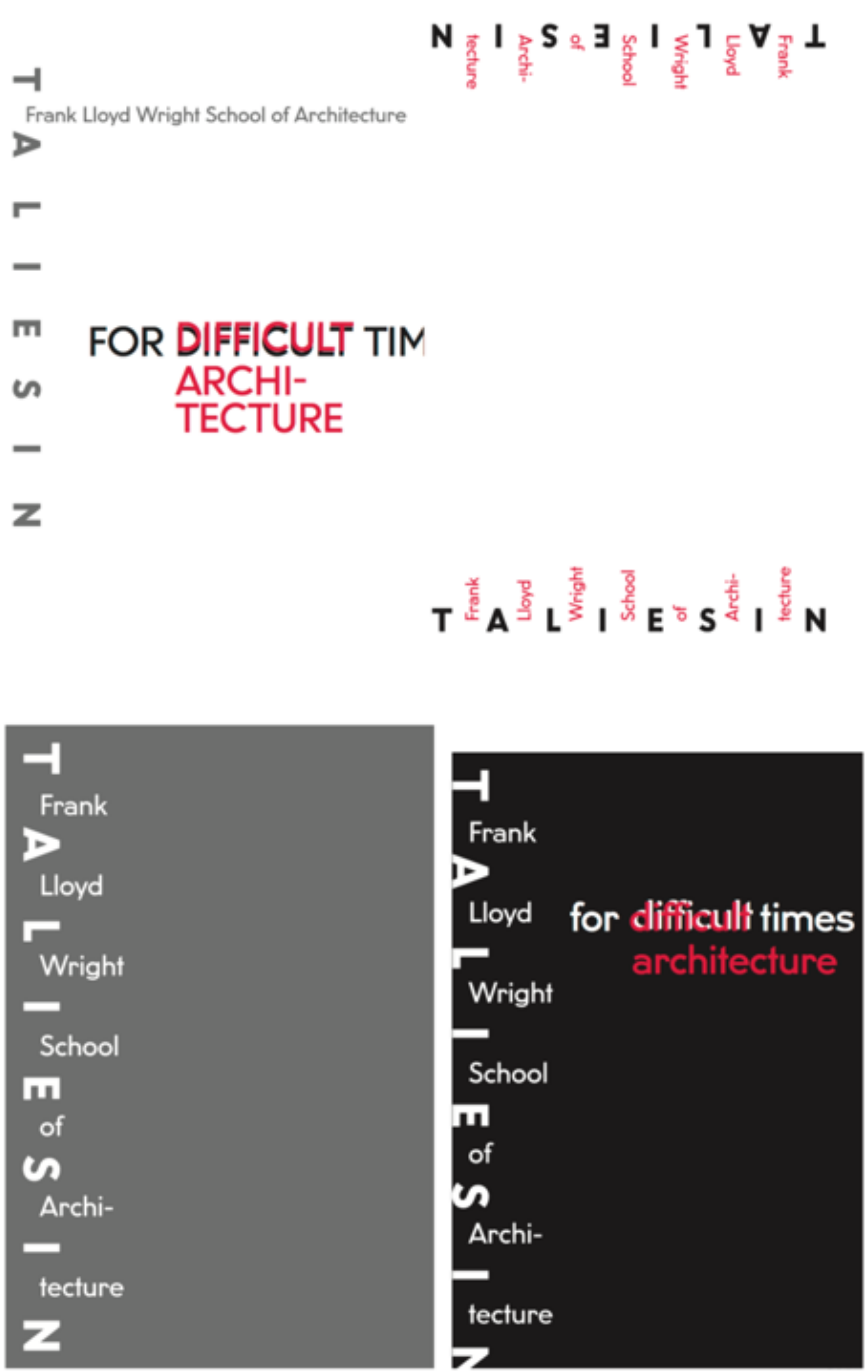
computer, interface, & designer

meta-process

We compiled submissions and feedback for PROCESS II via Slack. Here, we've sifted through the drafts and commentary, pulling back the curtain to highlight some early context for everyone's pieces.

neeta 1:35 AM

recently designed a poster for a lecture series at the frank lloyd wright school of architecture. the attached is my complete indesign document, from various stages of the project.


TALIESIN
Frank Lloyd Wright School of Architecture
FOR DIFFICULT TIM
ARCHI-
TECTURE
Frank Lloyd Wright School of Archi- tecture
for difficult times
architecture


TALIESIN
Frank Lloyd Wright School of Archi- tecture
Inaki Aba- los—Greg Lynn—Pa- tricia and John Pat- kau—Tatia- na Bilbao— Eric Owen Moss— Craig Hodgetts and Ming Fung—Da- vid Adjaye— Francine Houben
difficult archi- tecture for difficult times
architecture times
difficult
Frank Lloyd Wright School of Architecture
for difficult times
archi- tecture
inaki aba- los—greg lynn—pa- tricia and john pat- kau—tatia- na bilbao— eric owen moss— craig hodgetts and ming fung—da- vid adjaye— francine houben

Neeta Patel

neeta 1:35 AM

recently designed a poster for a lecture series at the frank lloyd wright school of architecture. the attached is my complete indesign document, from various stages of the project.

> page 17 (of 21) in this document is the final piece we printed—just to show that this is not necessarily an 'evolution' or 'forward progression' of ideas from primitive to final, but a series of nuanced ideas, some more formed than others.


TALIESIN
Frank Lloyd Wright School of Archi-tecture
patricia and john
January 12, 2017
patkau—
craig hodgetts and ming
January 26, 2017
fung—
eric owen
March 2, 2017—5:00 p.m.
moss—
March 16, 2017
francine houben
—tatiana bilbao—
March 29, 2017
michiel riedijk—
April 3, 2017
wolf d. prix—hani rashid and lise
April 6, 2017
anne couture—
April 13, 2017
david adjaye—
TBD
bjarke ingels—
TBD
difficult archi-tecture for difficult times
All lectures in the Taliesin Fo-rum, unless otherwise noted, begin at 7:30 p.m. in the Pavil-ion at Taliesin West, located at 12621 N. Frank Lloyd Wright Blvd., Scottsdale, Arizona 85259-2537.
Open to the public free of charge. Sponsored by the Graham Foundation for Ad-vanced Studies in the Fine Arts and The Río Salado Founda-tion. For more information please contact Matt Honer at TaliesinForum@taliesin.edu or 480 – 627 – 5349.

Neeta Patel

neeta 1:35 AM

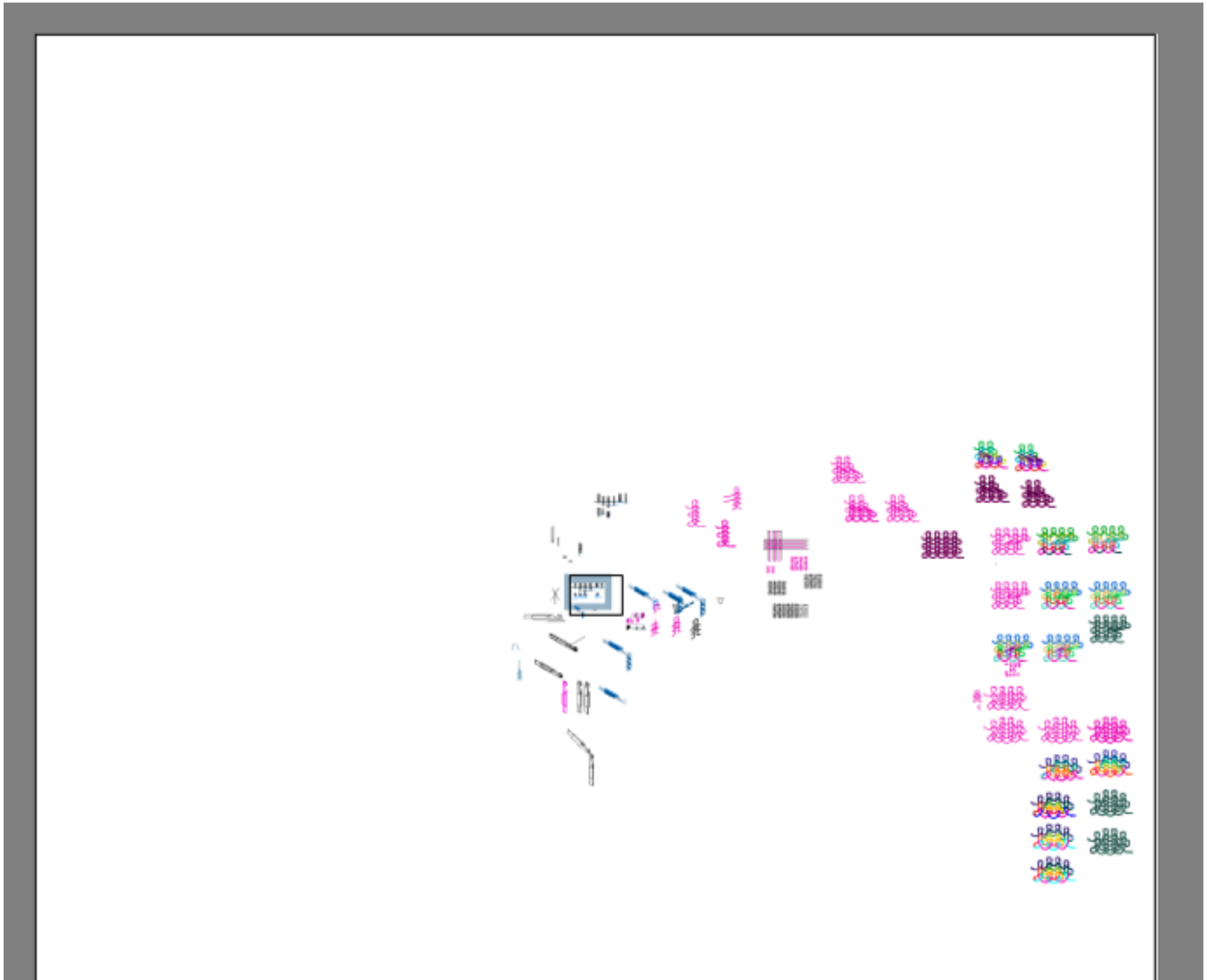
recently designed a poster for a lecture series at the frank lloyd wright school of architecture. the attached is my complete indesign document, from various stages of the project. (in my process, i duplicate the current working page every time i have a divergent thought / idea, to keep a record in case i decide a previous version was actually preferable). page 17 (of 21) in this document is the final piece we printed—just to show that this is not necessarily an 'evolution' or 'forward progression' of ideas from primitive to final, but a series of nuanced ideas, some more formed than others.

lea 4:20 PM

> (in my process, i duplicate the current working page every time i have a divergent thought / idea, to keep a record in case i decide a previous version was actually preferable)

I do this too!

Newer versions of Illustrator have bigger canvases and that is good for me.



kye 7:47 PM

> (in my process, i duplicate the current working page every time i have a divergent thought / idea, to keep a record in case i decide a previous version was actually preferable)

Yeah, I also do that in Photoshop / Procreate, but I duplicate the working layers and turn off their opacity instead of keeping them around on the page. Really wish these kinds of applications had undo trees instead of stacks.

👆 1

kye 7:47 PM

> (in my process, i duplicate the current working page every time i have a divergent thought / idea, to keep a record in case i decide a previous version was actually preferable)

Yeah, I also do that in Photoshop / Procreate, but I duplicate the working layers and turn off their opacity instead of keeping them around on the page. Really wish these kinds of applications had undo trees instead of stacks.

👆 1

beiser 8:03 PM

(the distinction between undo/redo and version control is a historical artifact that needs to be remedied)

kye 8:06 PM

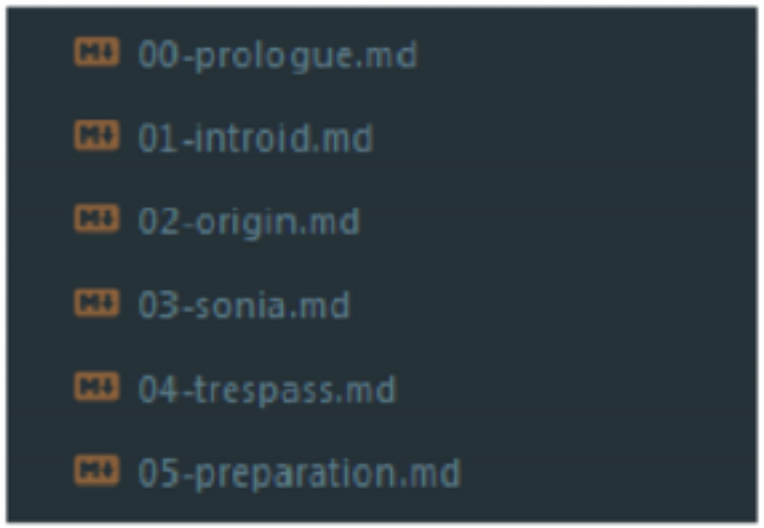
Yes! Was just thinking about how undo trees at the diff level are just version control, and version control at the sentence / "primitive" level is just undo trees.

Diana’s piece on version control:

I am currently writing an as-of-unnamed science fantasy novel using Sublime Text 3 and Markdown.

I am currently writing an as-of-unnamed science fantasy novel using Sublime Text 3 and Markdown.

Every chapter is a file...



....and the whole manuscript is put together using a bash script that concatenates the chapters in name order, and then outputs a .docx file.

The reason why I'm using this setup is to make sure that all of the source text is in plaintext, primarily so that it can work with version control. The entire novel is backed by a Mercurial repository. This generally works pretty well.

(it's reassuring to have backups to reference!)

Diana Liao (highlights added)

Commits themselves aren't terribly useful for diffing for the fact that they are based on lines, which for a novel ends up being entire paragraphs. (Also, my commits are just a big mess.) I don't use them to compare differences, just to have a backup of prose that I wrote and might've cut, in case I want to revisit it.

In which I edit too many chapters in one sitting:

Files changed (15)

+4	-1	M	03-sonia.md
+3	-0	M	09-investigation.md
+2	-31	M	14n2-hideout.md
+44	-0	M	16x-town.md
+103	-2	M	21-awakening.md
+7	-43	M	21x-inevitability.md
+116	-5	M	22-celebratory.md
+117	-0	A	22x-initial.md
+3	-1	M	24-lone.md
+0	-0	M	build/all.docx
+789	-111	M	build/all.md

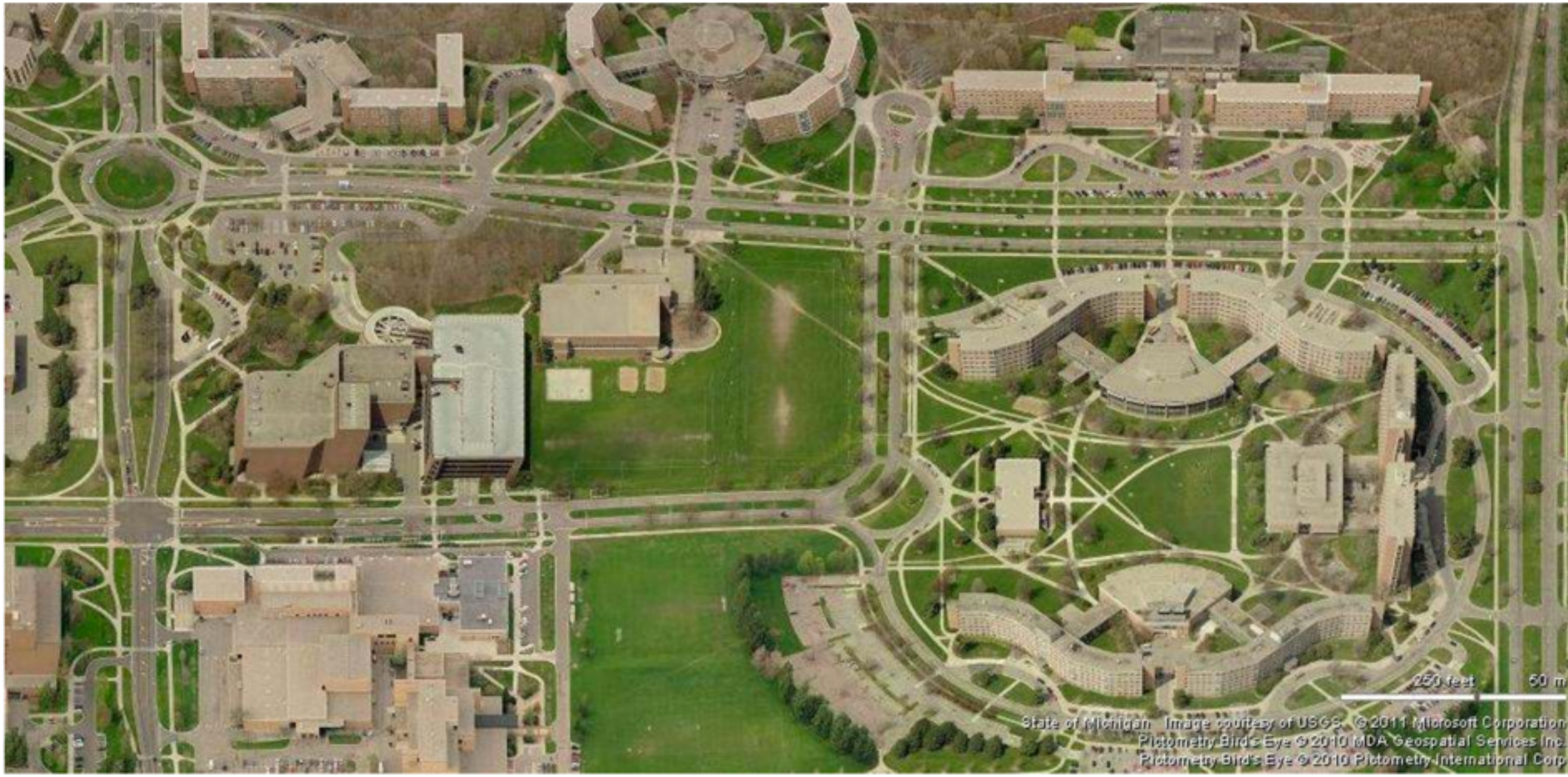
Diana Liao (highlights added)

oh, here's the current build script

```
rm -rf build
mkdir -p build

# base concatenation
for each in *.md; do (cat $each; printf "\n\n---\n\n") >>
    build/base.md; done
# cat *.md > build/all.md

# convert all single returns into doubles so that pandoc doesn't complain
cat build/base.md | awk -v RS="" '{gsub (/\n\n/,"\n")}1' |
    awk -v RS="" '{gsub (/\n/,"\n\n")}1' >> build/all.md;

# create the .docx file
pandoc -f markdown_github -t docx -o build/all.docx build/all.md
```

We stumbled on *desire paths* in creative interfaces.

http://99percentinvisible.org/article/least-resistance-desire-paths-can-lead-better-design/



Emergent desire paths turned into sidewalks over time at Michigan State University

http://99percentinvisible.org/article/least-resistance-desire-paths-can-lead-better-design/

Case study: how can we turn desire paths into sidewalks?

This idea is closely related to Michael Nielsen's ideas on reifying hidden representations and deep principles about the world in interfaces. See "Thought as a Technology."

Chris Beiser

7:26 PM Jun 3

Resolve

I’ve also been thinking about writing process—I sometimes get stuck on a single paragraph, and the best way to get unstuck is to write many ‘wrong’ attempts at the paragraph as fast as possible, so that I have material to work from and combine; tools don’t tend to support that one much.


First Draft of the Revolution
A PARIS
J. Nochez inv. et sculp.

The act of writing inevitably changes what you want to say…

[It's] a process, not just of setting down a concept already fully formed in the mind, but of **discovering what it is you meant in the first place.**

First Draft of the Revolution
Emily Short & Liza Daly

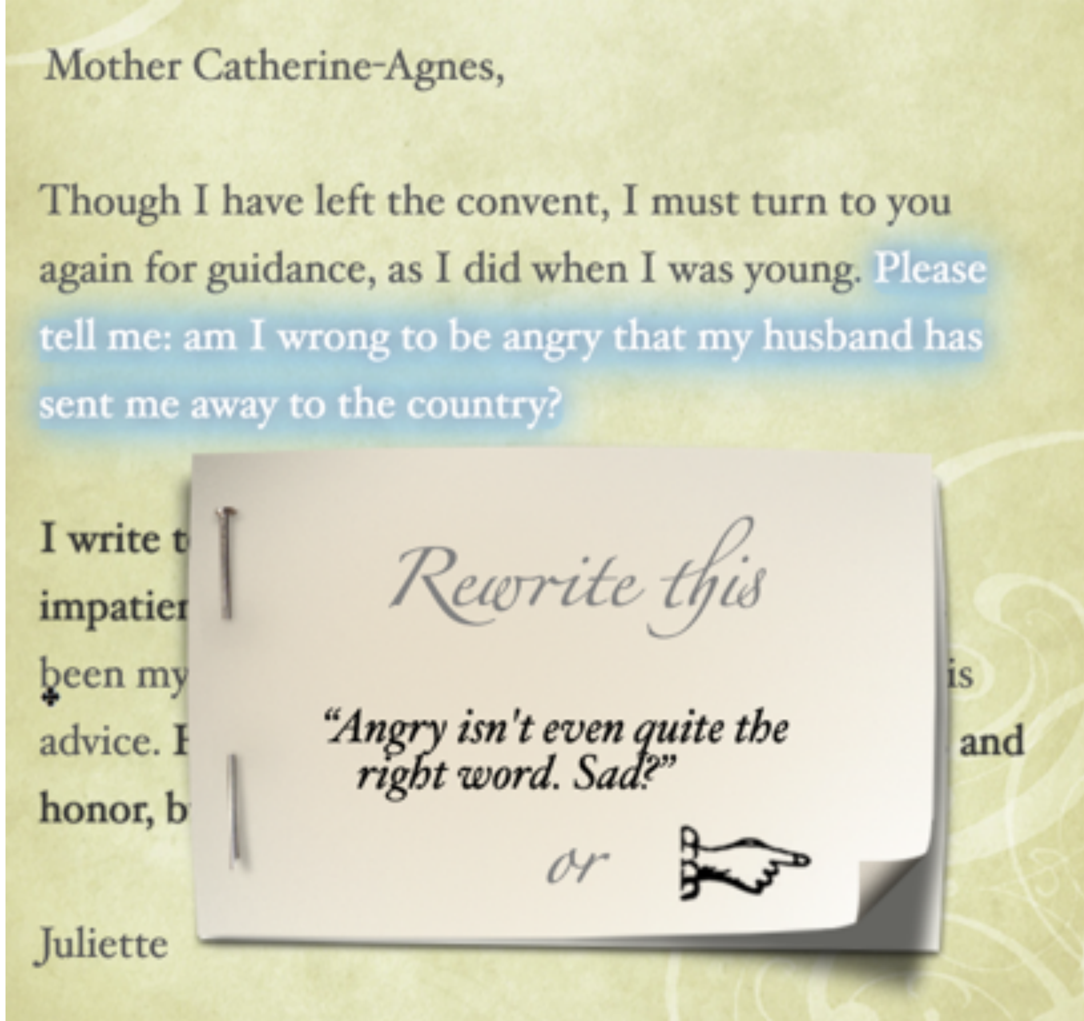
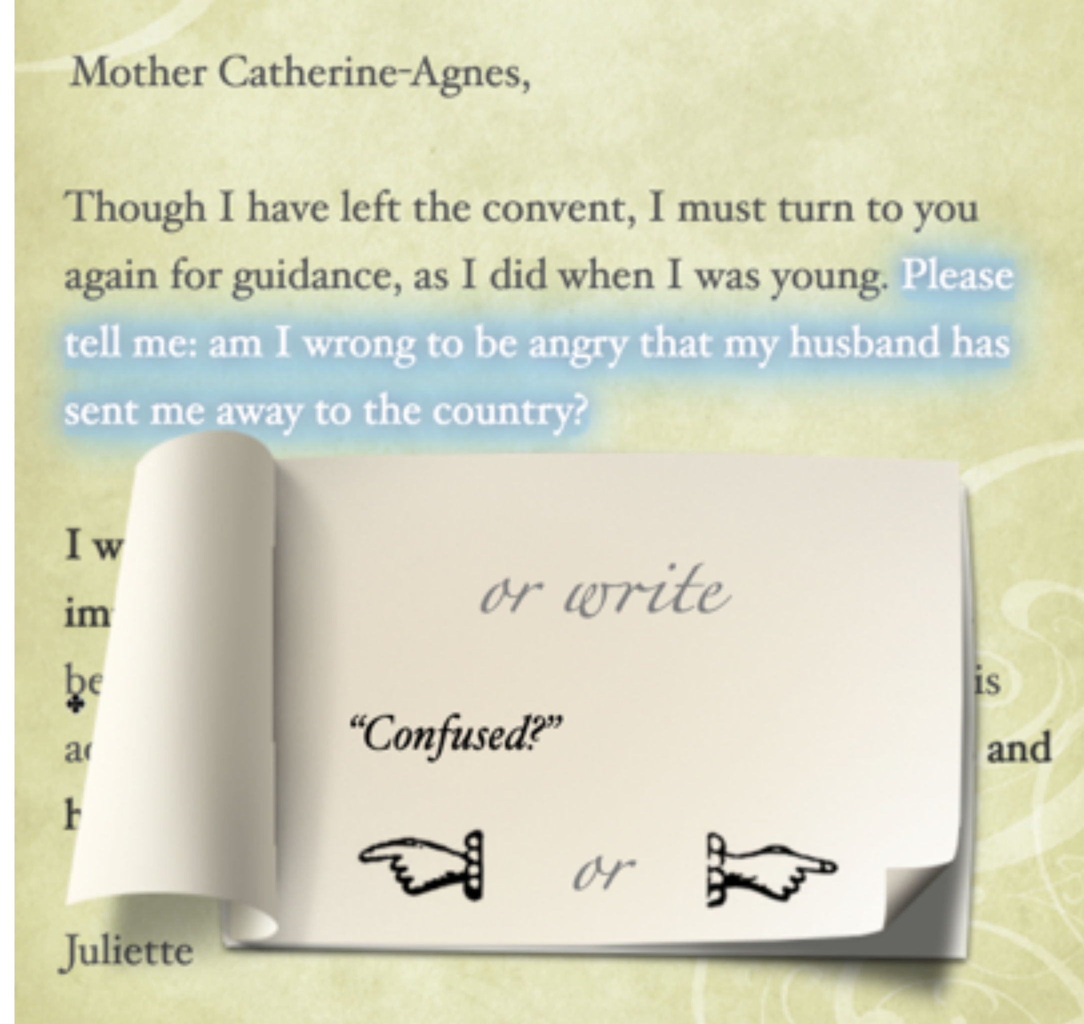
Dauphiné, July 1788

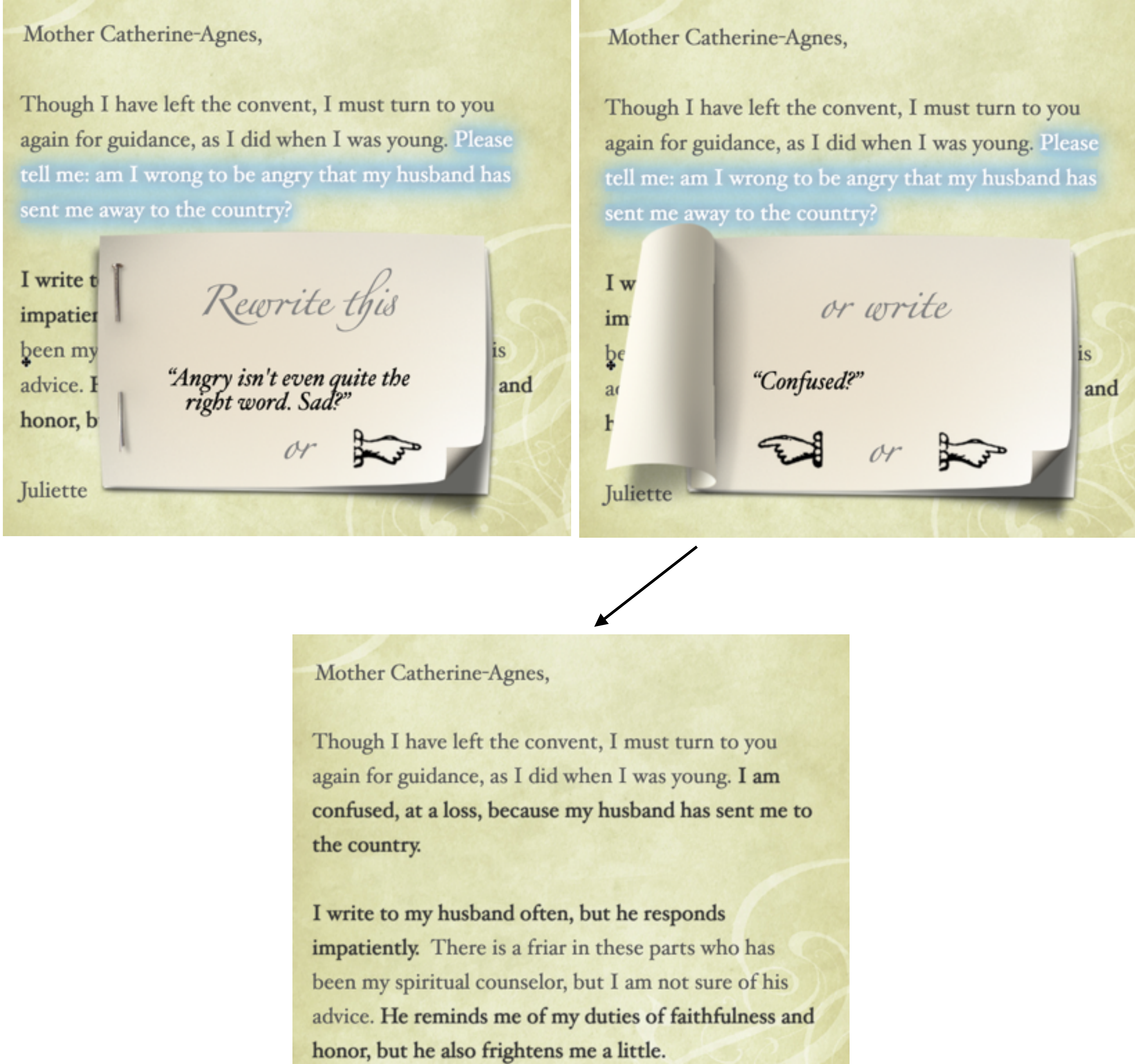
Mother Catherine-Agnes,

Though I have left the convent, I must turn to you again for guidance, as I did when I was young. **Please tell me: am I wrong to be angry that my husband has sent me away to the country?**

I write to my husband often, but he responds impatiently. There is a friar in these parts who has been my spiritual counselor, but I am not sure of his advice. **He reminds me of my duties of faithfulness and honor, but he also frightens me a little.**

Juliette


Mother Catherine-Agnes,
Though I have left the convent, I must turn to you again for guidance, as I did when I was young. Please tell me: am I wrong to be angry that my husband has sent me away to the country?
Rewrite this
“Angry isn't even quite the right word. Sad?”
or
Juliette

Mother Catherine-Agnes,
Though I have left the convent, I must turn to you again for guidance, as I did when I was young. Please tell me: am I wrong to be angry that my husband has sent me away to the country?
or write
“Confused?”
or
Juliette


Mother Catherine-Agnes,
Though I have left the convent, I must turn to you again for guidance, as I did when I was young. Please tell me: am I wrong to be angry that my husband has sent me away to the country?
Rewrite this
"Angry isn't even quite the right word. Sad?"
or
Juliette
Mother Catherine-Agnes,
Though I have left the convent, I must turn to you again for guidance, as I did when I was young. Please tell me: am I wrong to be angry that my husband has sent me away to the country?
or write
"Confused?"
or
Juliette
Mother Catherine-Agnes,
Though I have left the convent, I must turn to you again for guidance, as I did when I was young. I am confused, at a loss, because my husband has sent me to the country.
I write to my husband often, but he responds impatiently. There is a friar in these parts who has been my spiritual counselor, but I am not sure of his advice. He reminds me of my duties of faithfulness and honor, but he also frightens me a little.

Though I have left the convent, I must turn to you again for guidance, as I did when I was young. **I am confused, at a loss, because my husband has sent me to the country.**

I write to my husband often, but he responds impatiently. There is a friar in these parts who has been my spiritual counselor, but I am not sure of his advice. He reminds me of my duties of faithfulness and honor, but he also frightens me a little.

Juliette

Rewrite this

"I will have to be more specific."

Though I have left the convent, I must turn to you again for guidance, as I did when I was young. **I am confused, at a loss, because my husband has sent me to the country.**

I write to my husband often, but he responds impatiently. There is a friar in these parts who has been my spiritual counselor, but I am not sure of his advice. He reminds me of my duties of faithfulness and honor, but he also frightens me a little.

Juliette

Rewrite this

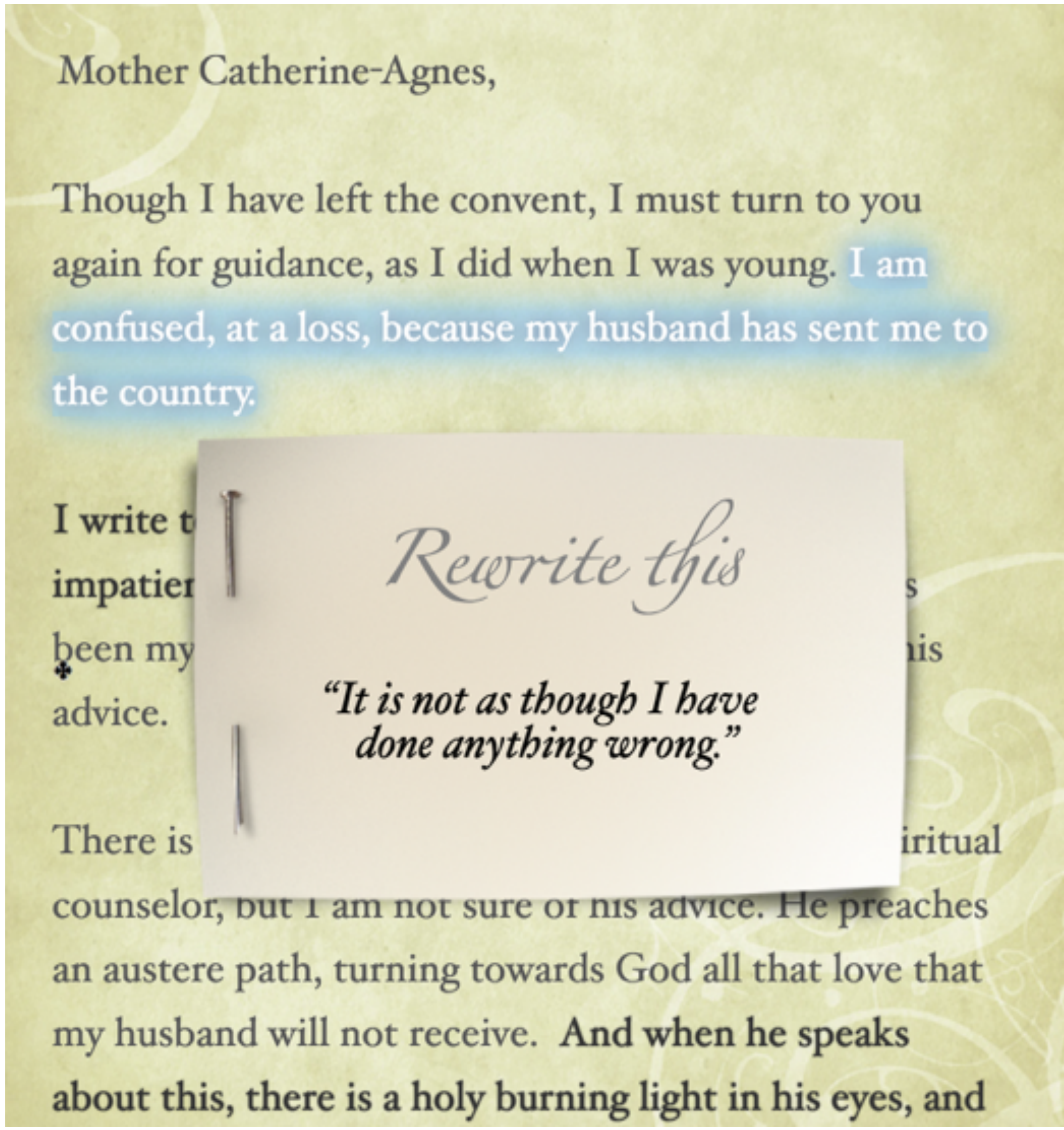
"I will have to be more specific."

There is a friar in these parts who has been my spiritual counselor, but I am not sure of his advice. He preaches an austere path, turning towards God all that love that my husband will not receive. **And when he speaks about this, there is a holy burning light in his eyes, and he clasps my hands in his; and I feel the strength of the spirit flow into me and know that I am capable of anything.** But when he is gone, my despair is so much blacker, for I have seen what true virtue must be, and how far I am away from it.

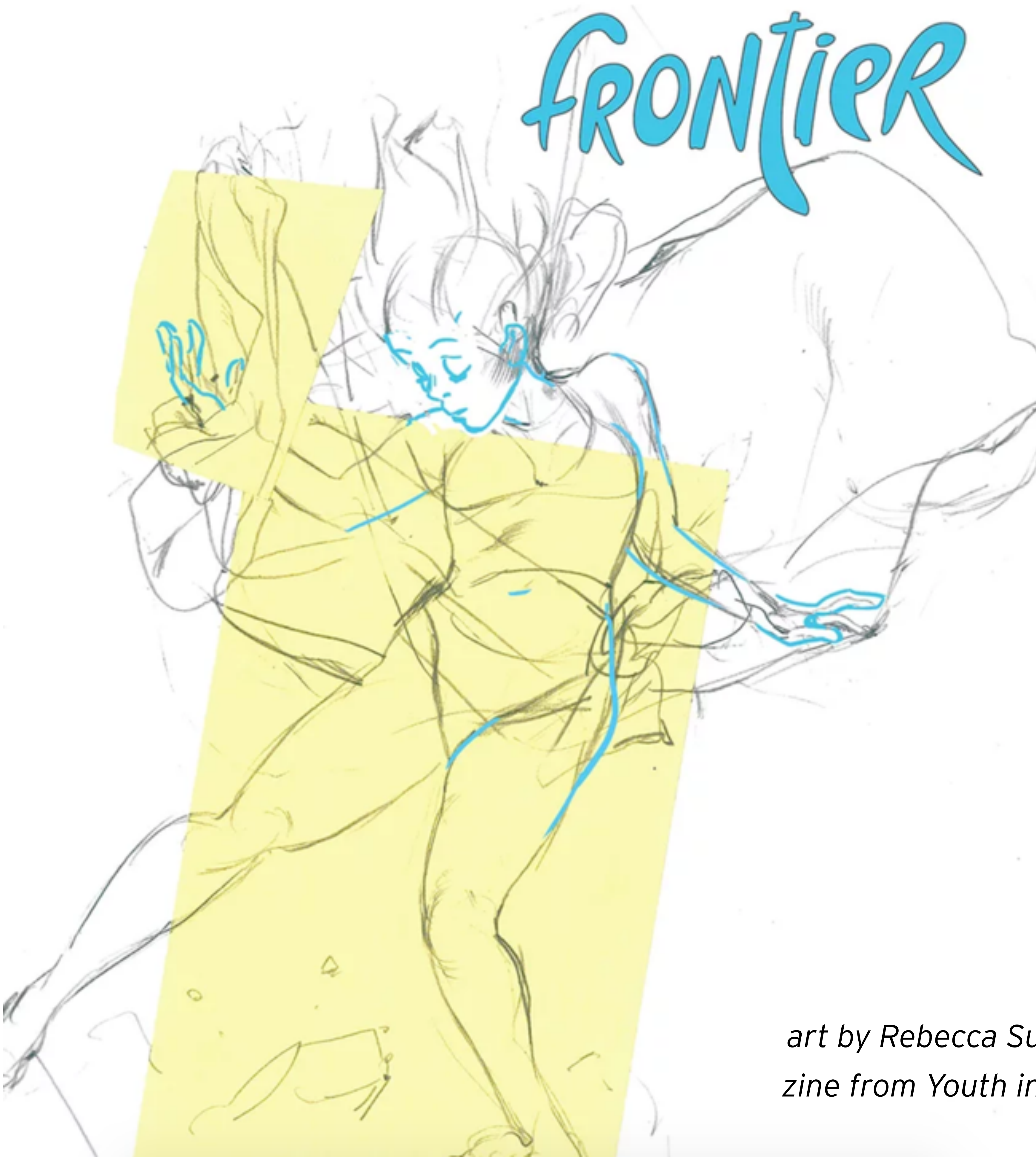
I am a little afraid of the friar too. **His zealous passion for good makes him angry against the magic-using ranks; he says that they have not been wise with their gift, and that they betray the poor.**

Juliette

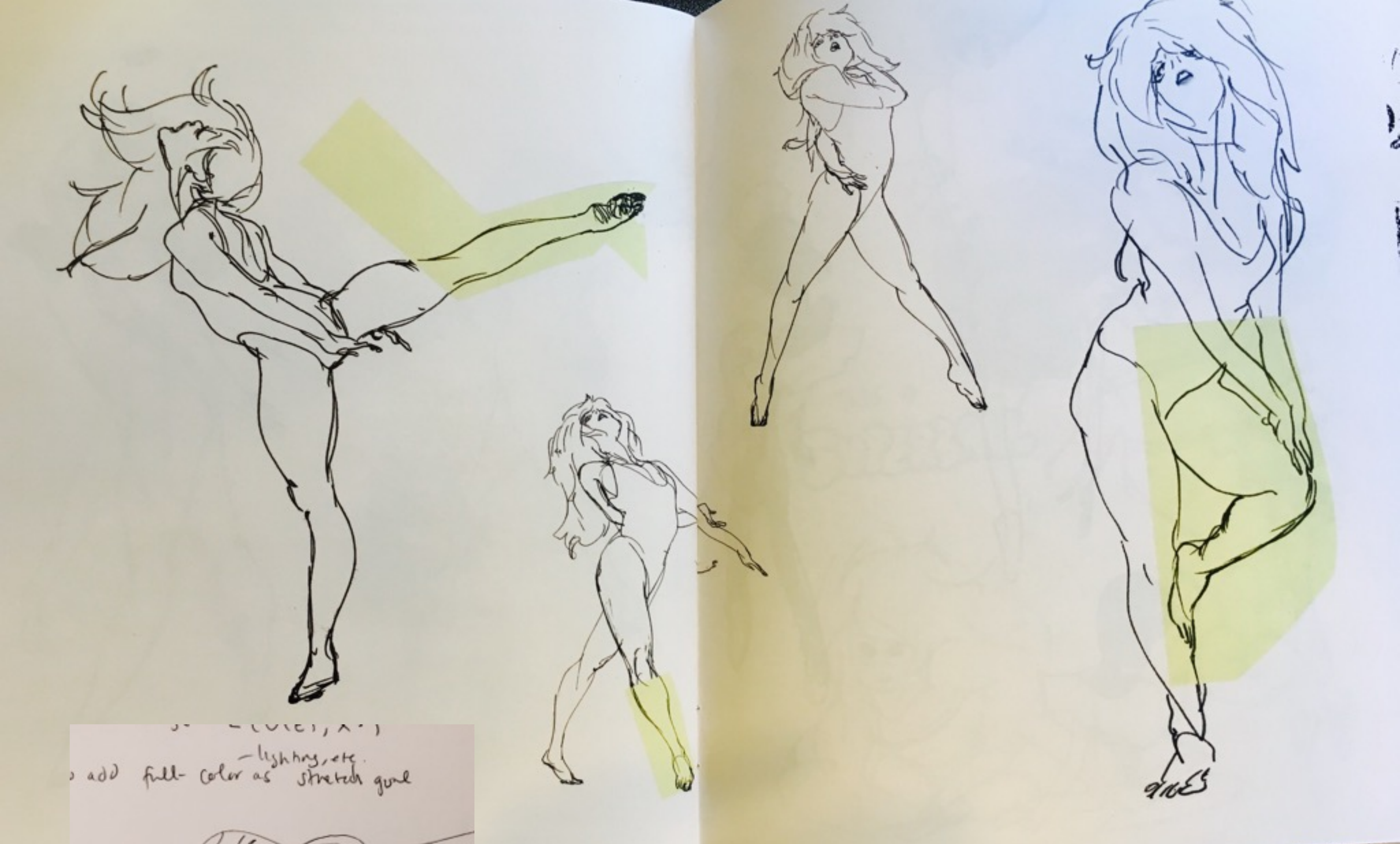
Rewriting one part helps you realize what you meant to say elsewhere!



Two more case studies:
how can we turn desire paths
into sidewalks?



art by Rebecca Sugar
zine from Youth in Decline



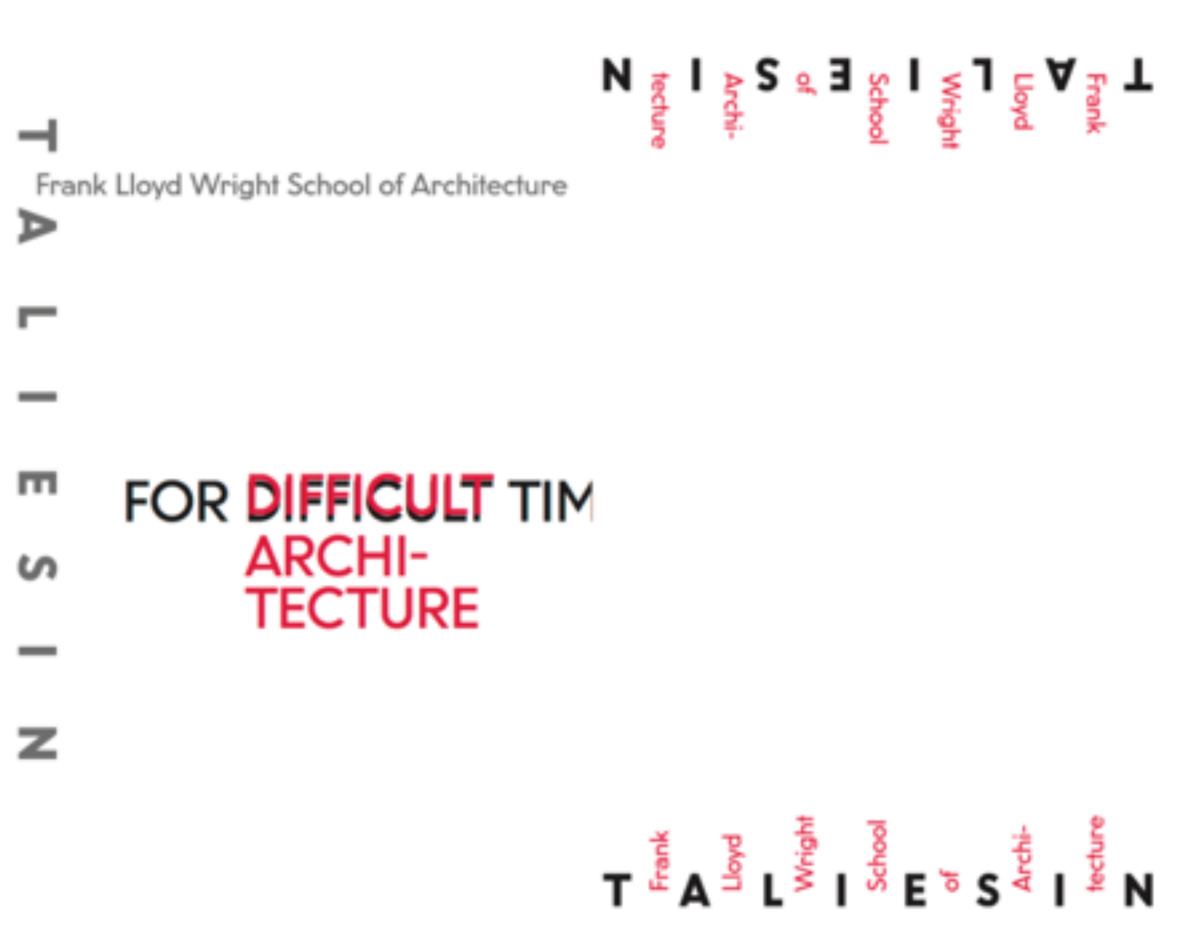
top: art by Rebecca Sugar

lea 4:20 PM

> (in my process, i duplicate the current working page every time i have a divergent thought / idea, to keep a record in case i decide a previous version was actually preferable)

I do this too!

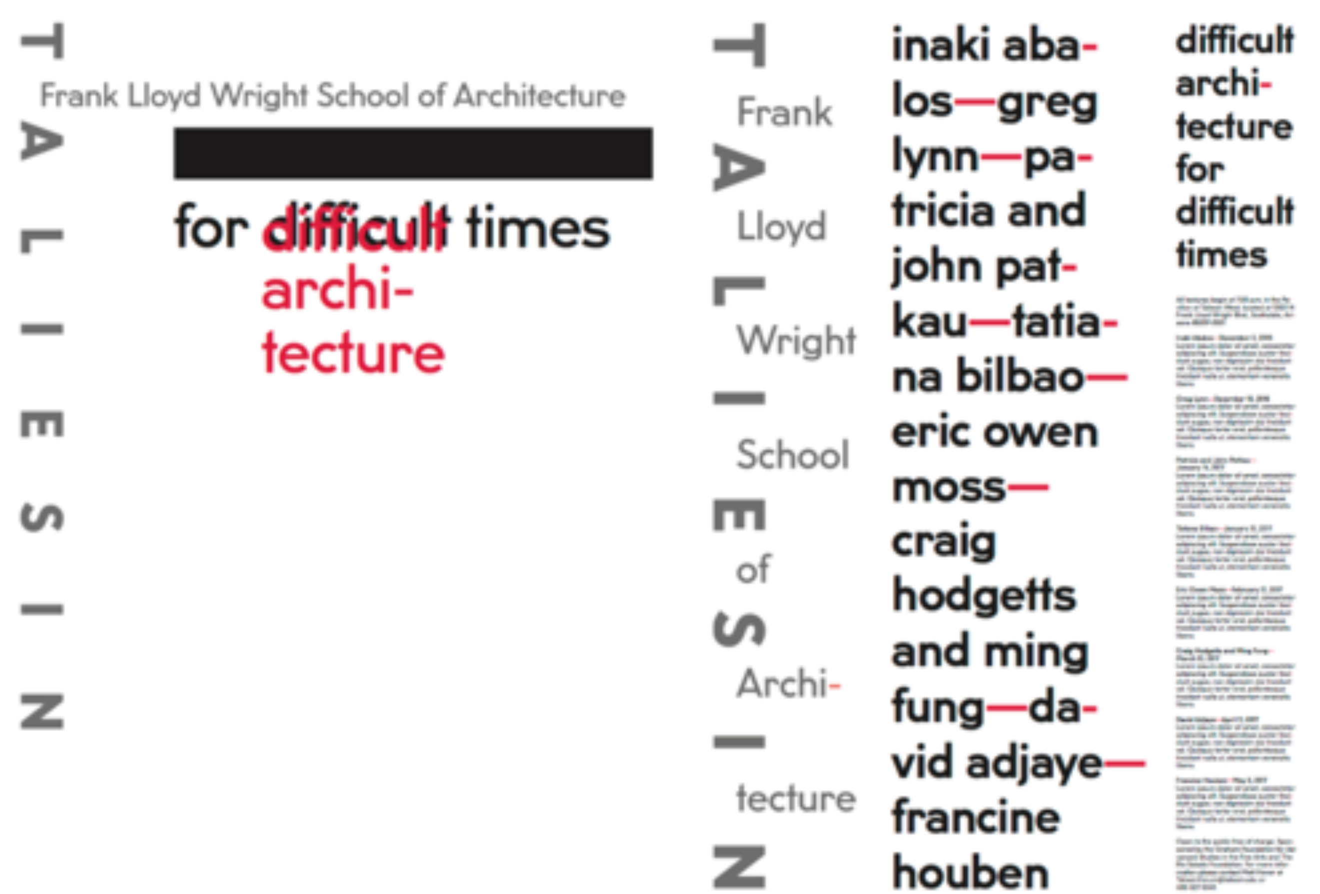
Newer versions of Illustrator have bigger canvases and that is good for me.


TALIESIN
Frank Lloyd Wright School of Architecture
FOR DIFFICULT TIM
ARCHI-
TECTURE
T Frank A Lloyd L Wright I School E of S Archi- I tecture N


T Frank A Lloyd L Wright I School E of S Archi- I tecture N

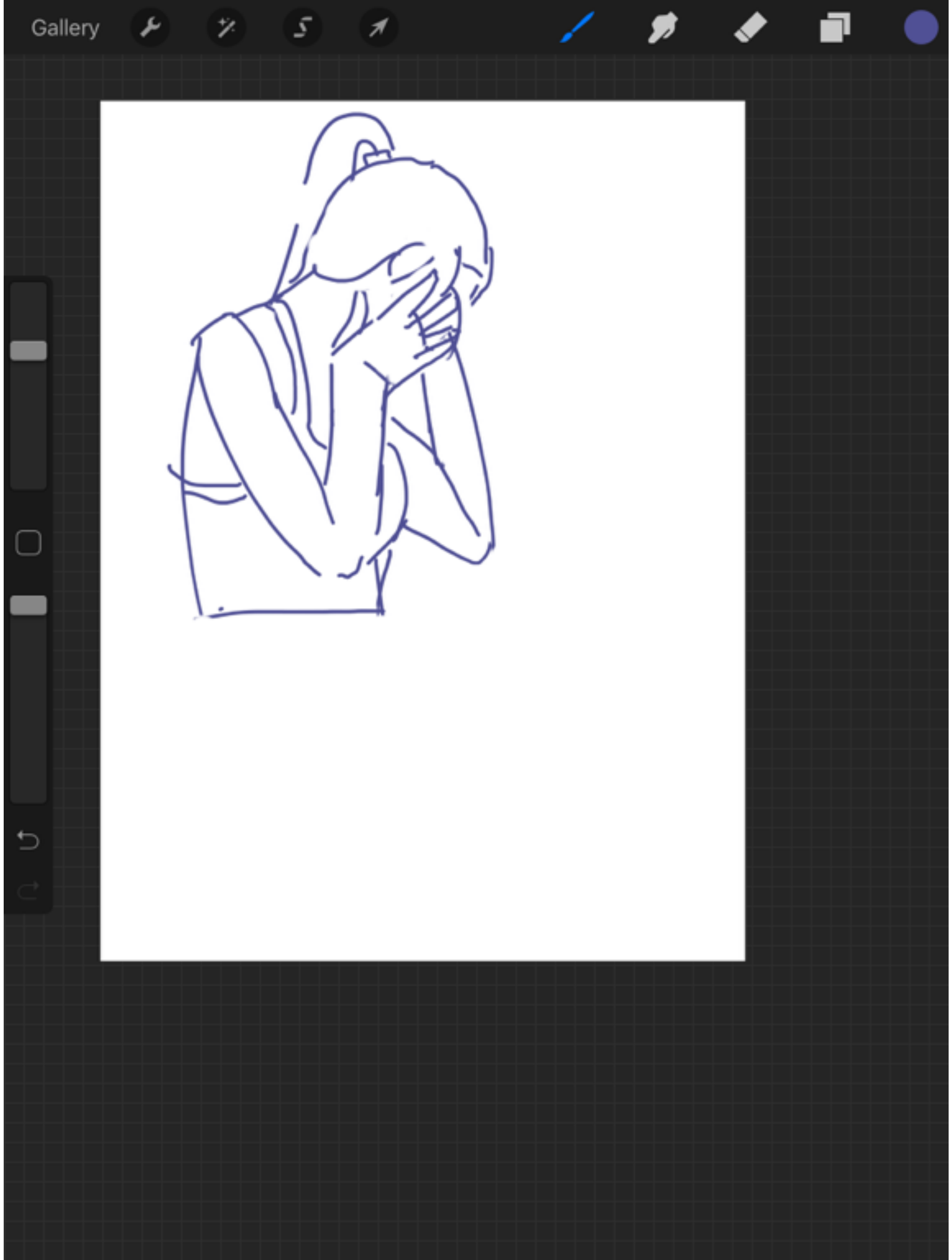

T Frank A Lloyd L Wright I School E of S Archi- I tecture N
for difficult times
architecture

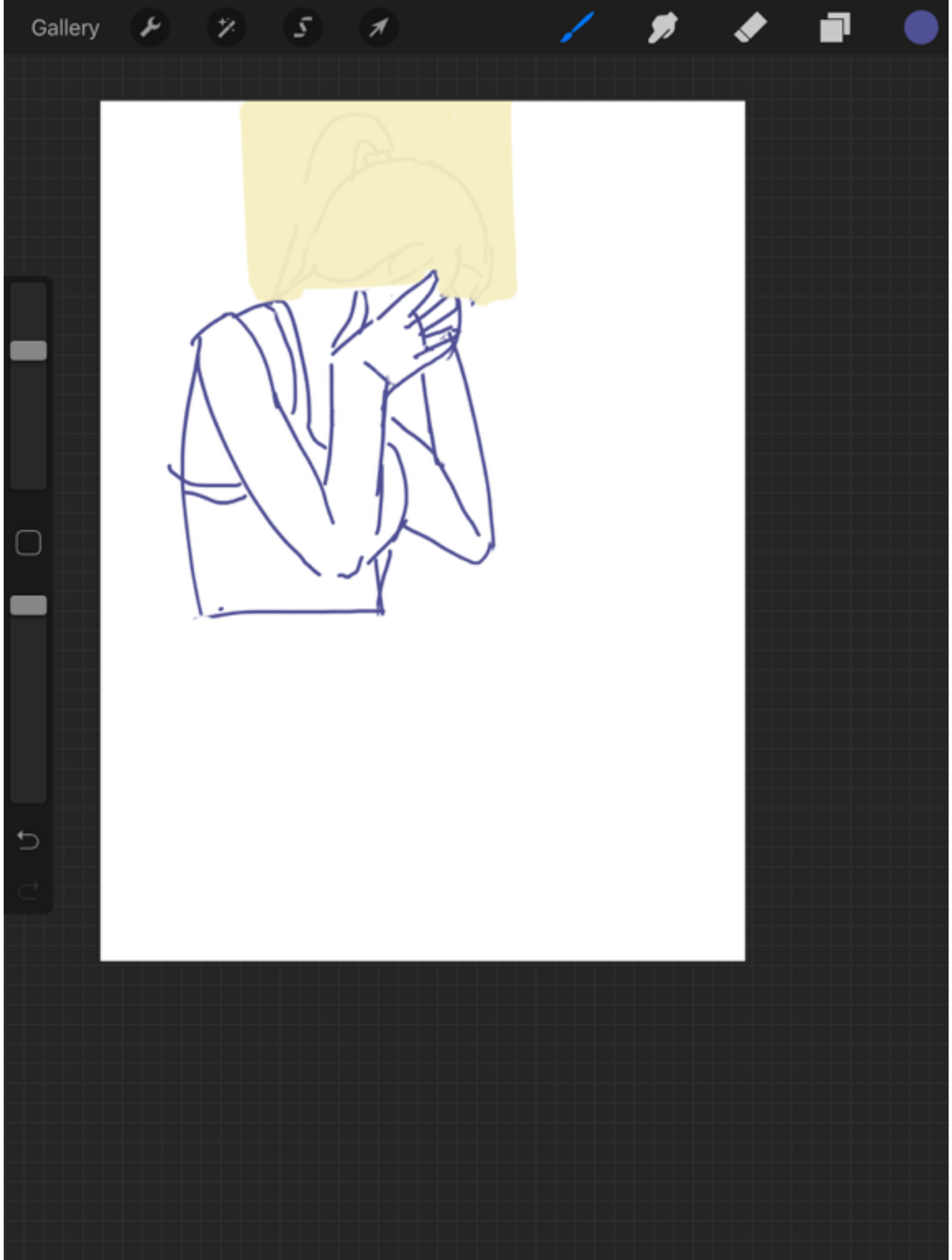

T Frank A Lloyd L Wright I School E of S Archi- I tecture N
Inaki Abalos—Greg Lynn—Patricia and John Patkau—Tatiana Bilbao—Eric Owen Moss—Craig Hodgetts and Ming Fung—David Adjaye—Francine Houben
difficult architecture for difficult times
architecture times
difficult
difficult
T Frank A Lloyd L Wright I School E of S Archi- I tecture N

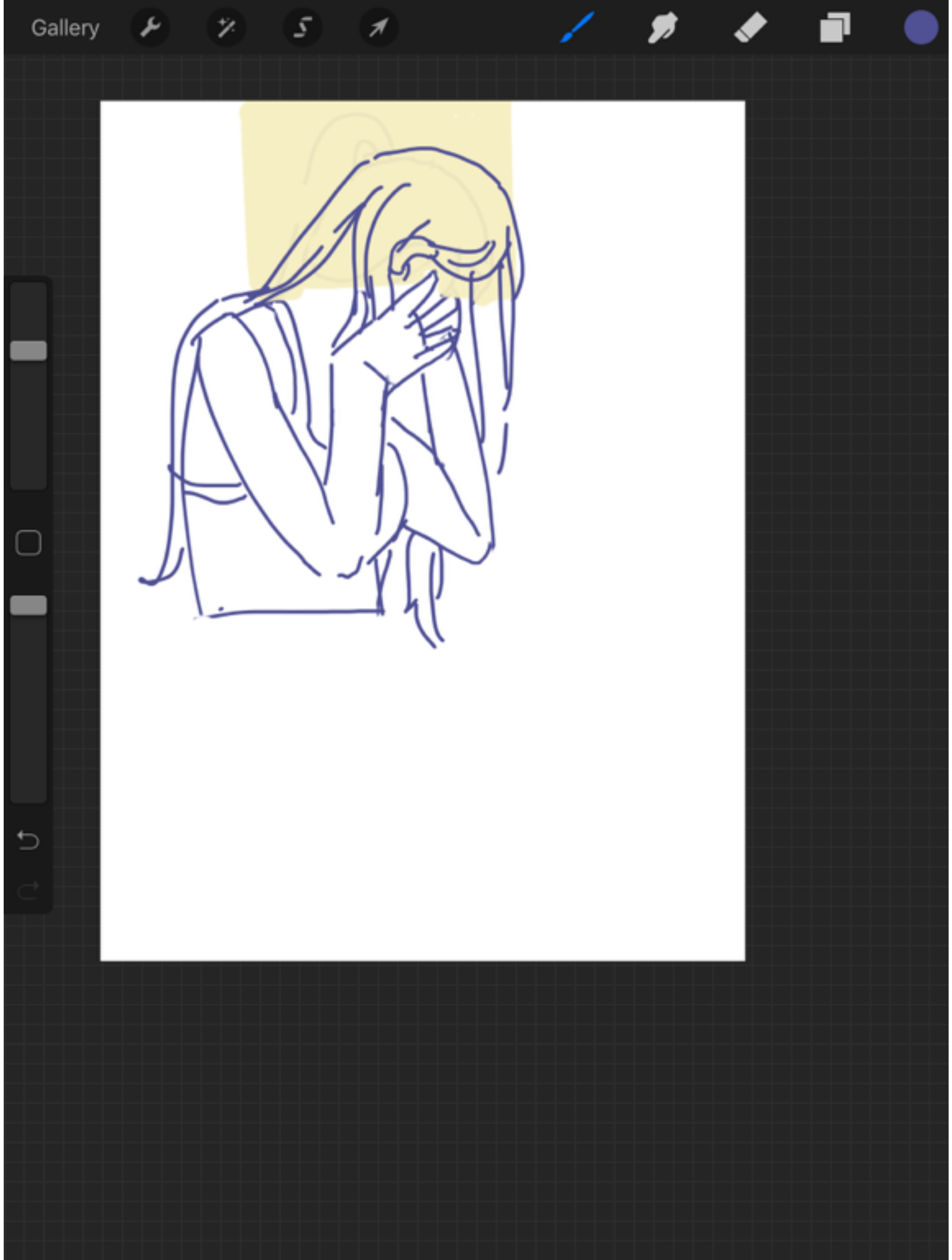

TALIESIN
Frank Lloyd Wright School of Architecture
for difficult times
archi-
tecture
T Frank A Lloyd L Wright I School E of S Archi- I tecture N
inaki abalos—greg lynn—patricia and john patkau—tatiana bilbao—eric owen moss—craig hodgetts and ming fung—david adjaye—francine houben
difficult architecture for difficult times

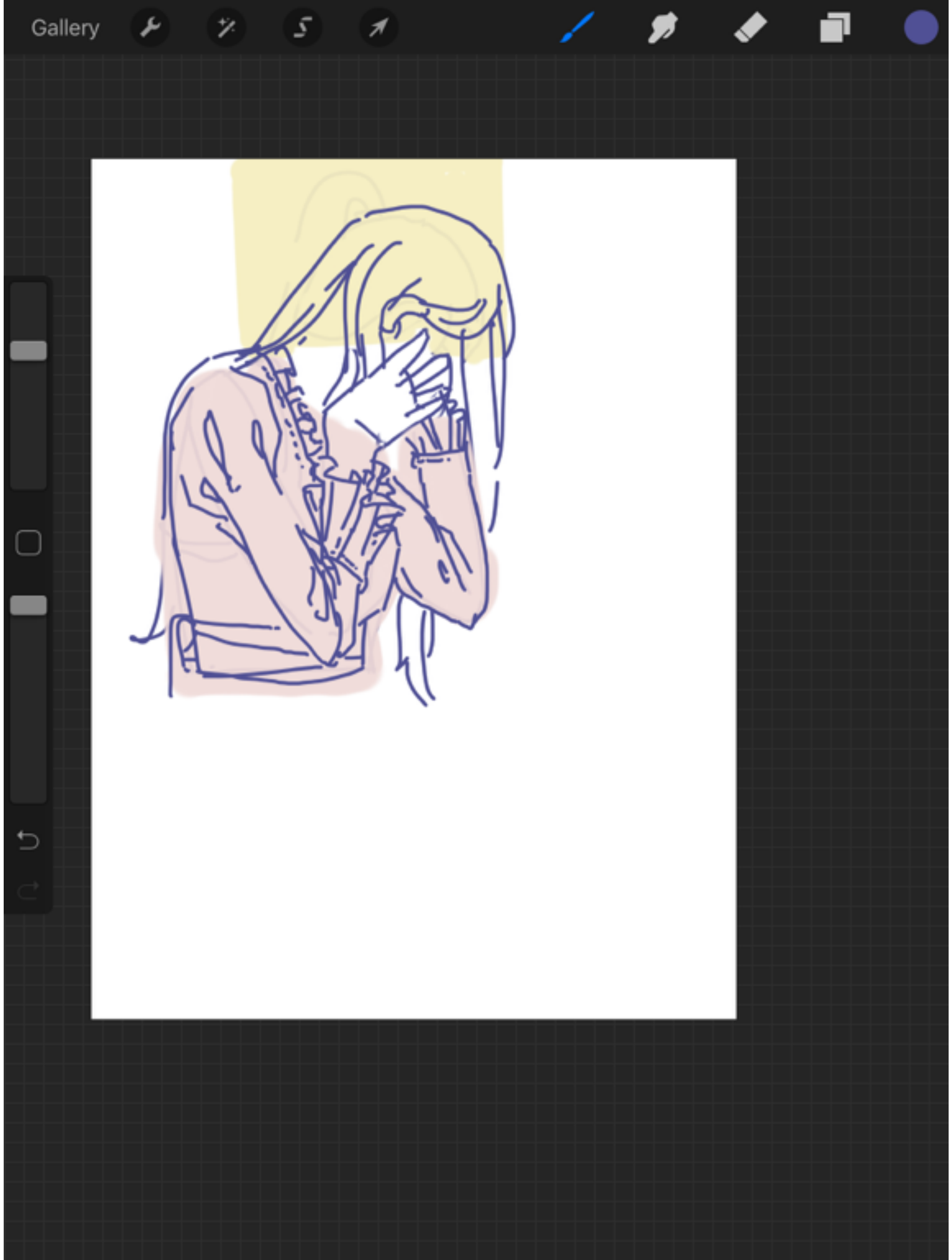
Neeta Patel

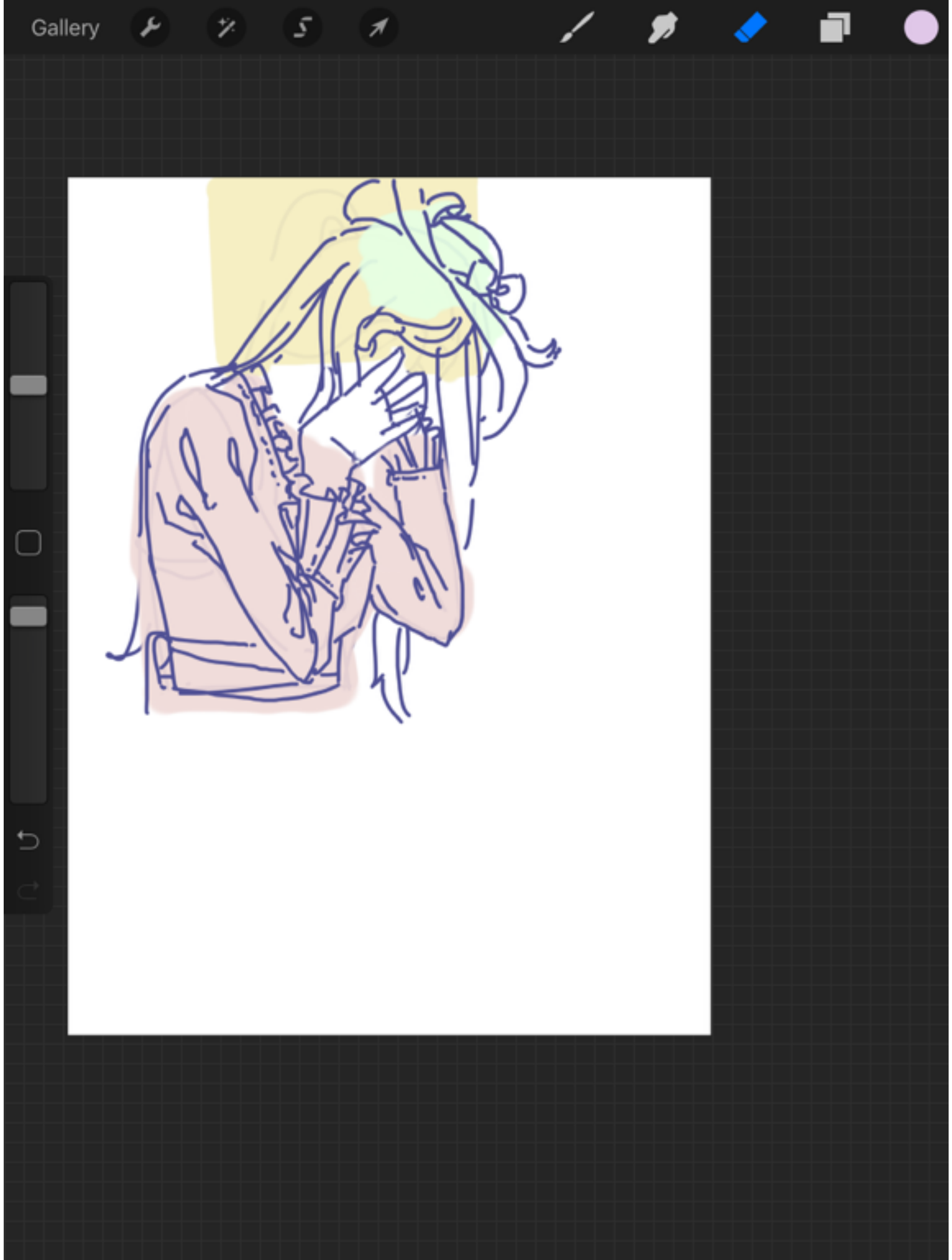
How can we enable artists to navigate alternate pasts and futures, to more easily explore regions and versions of their work?

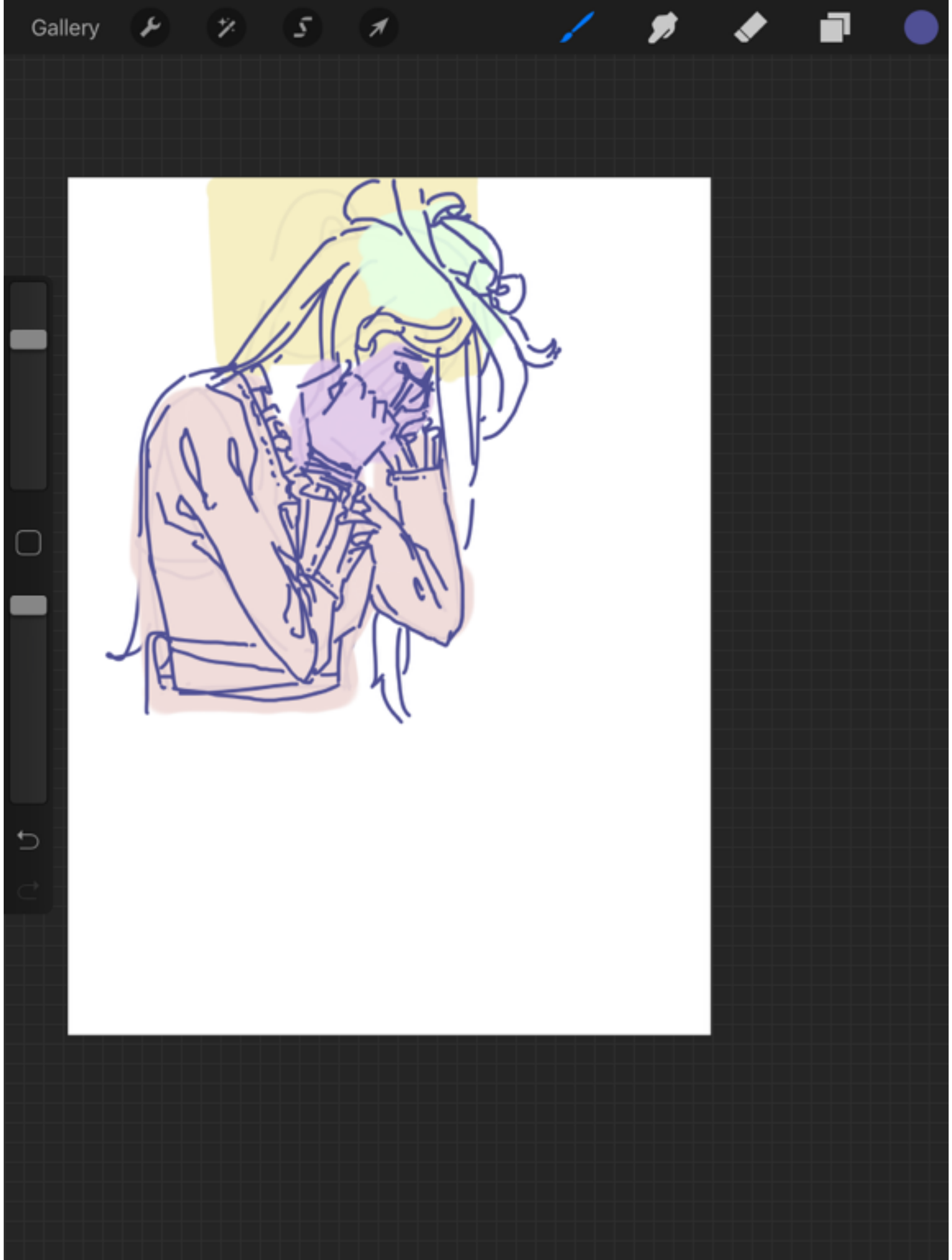

Gallery

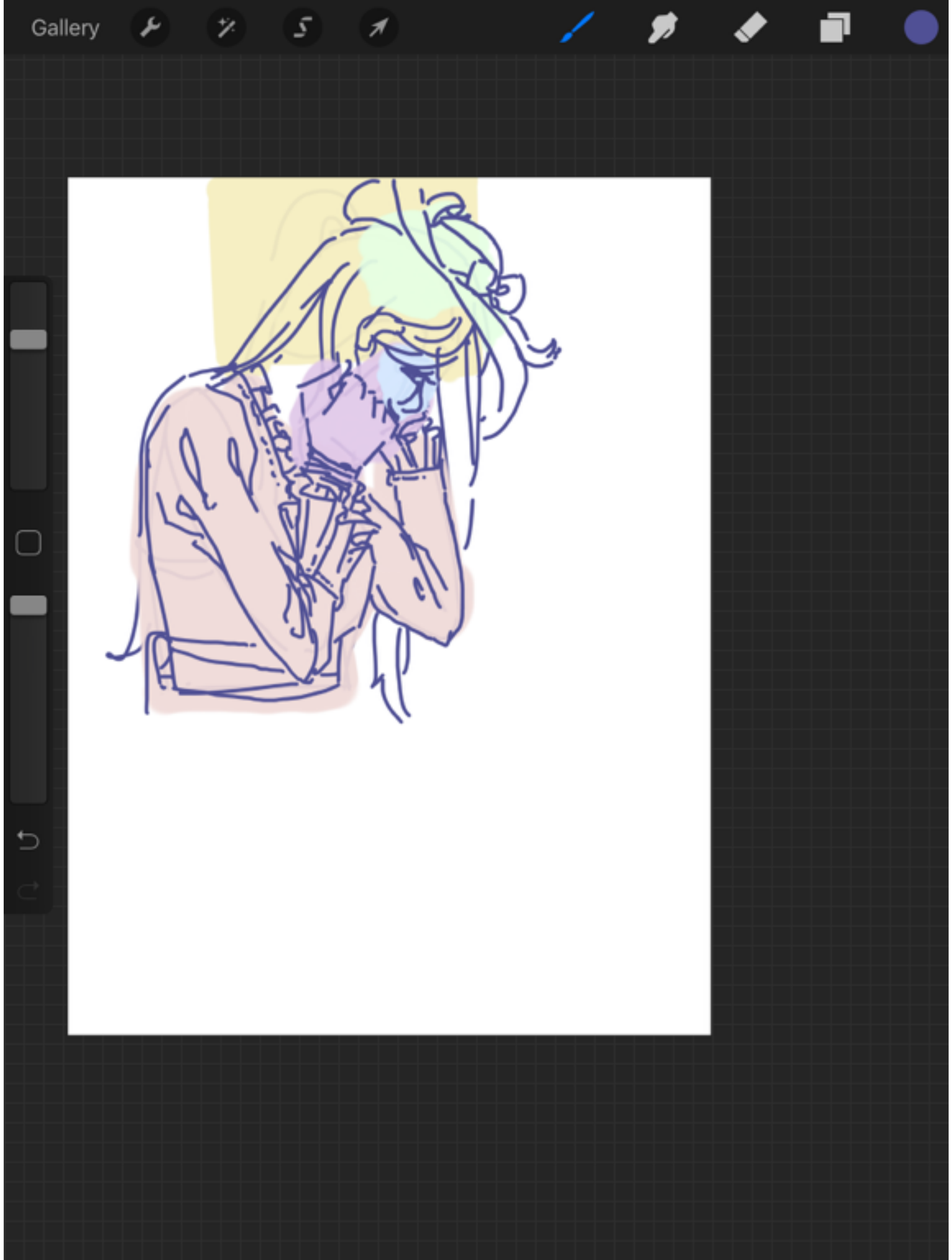

Gallery

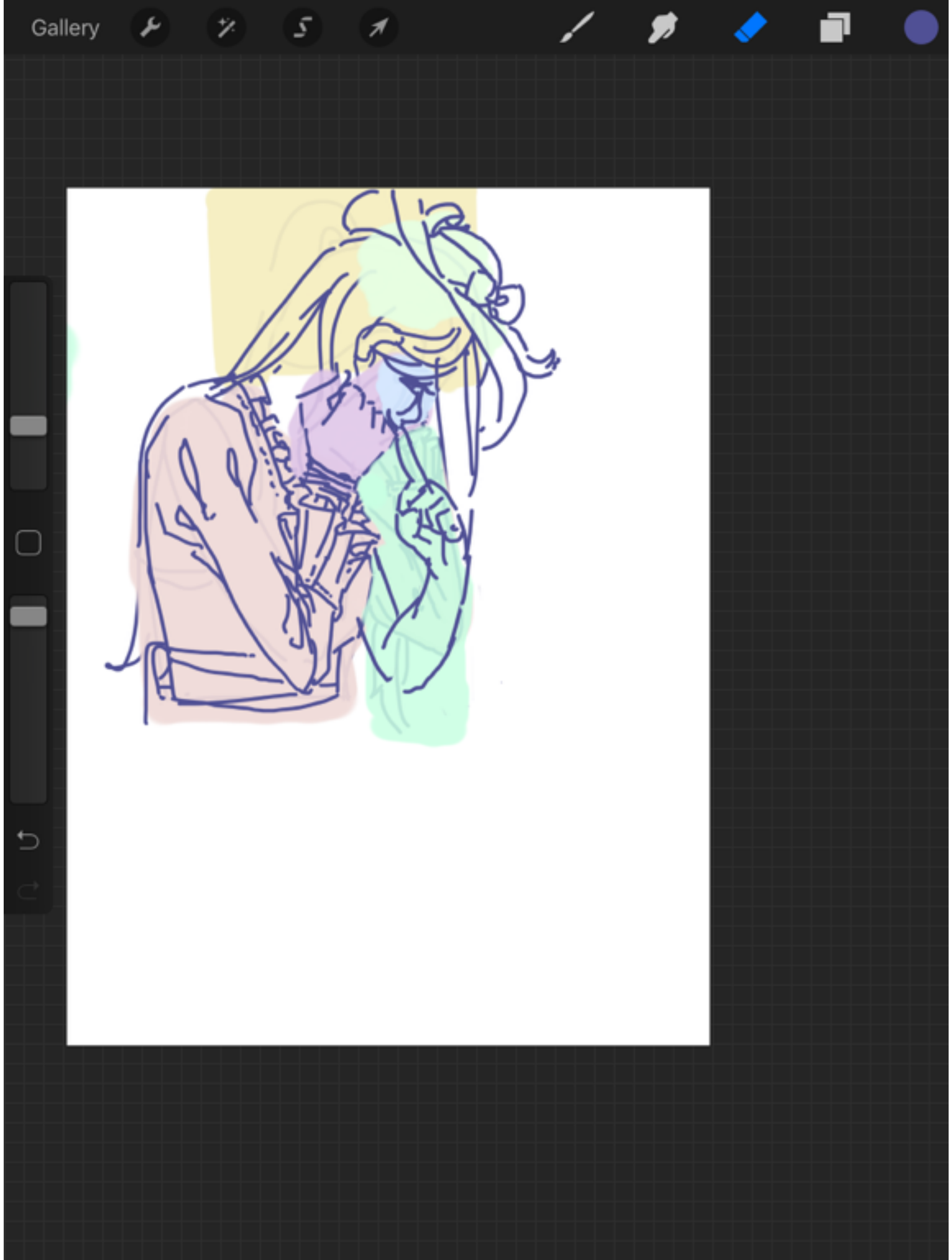

Gallery

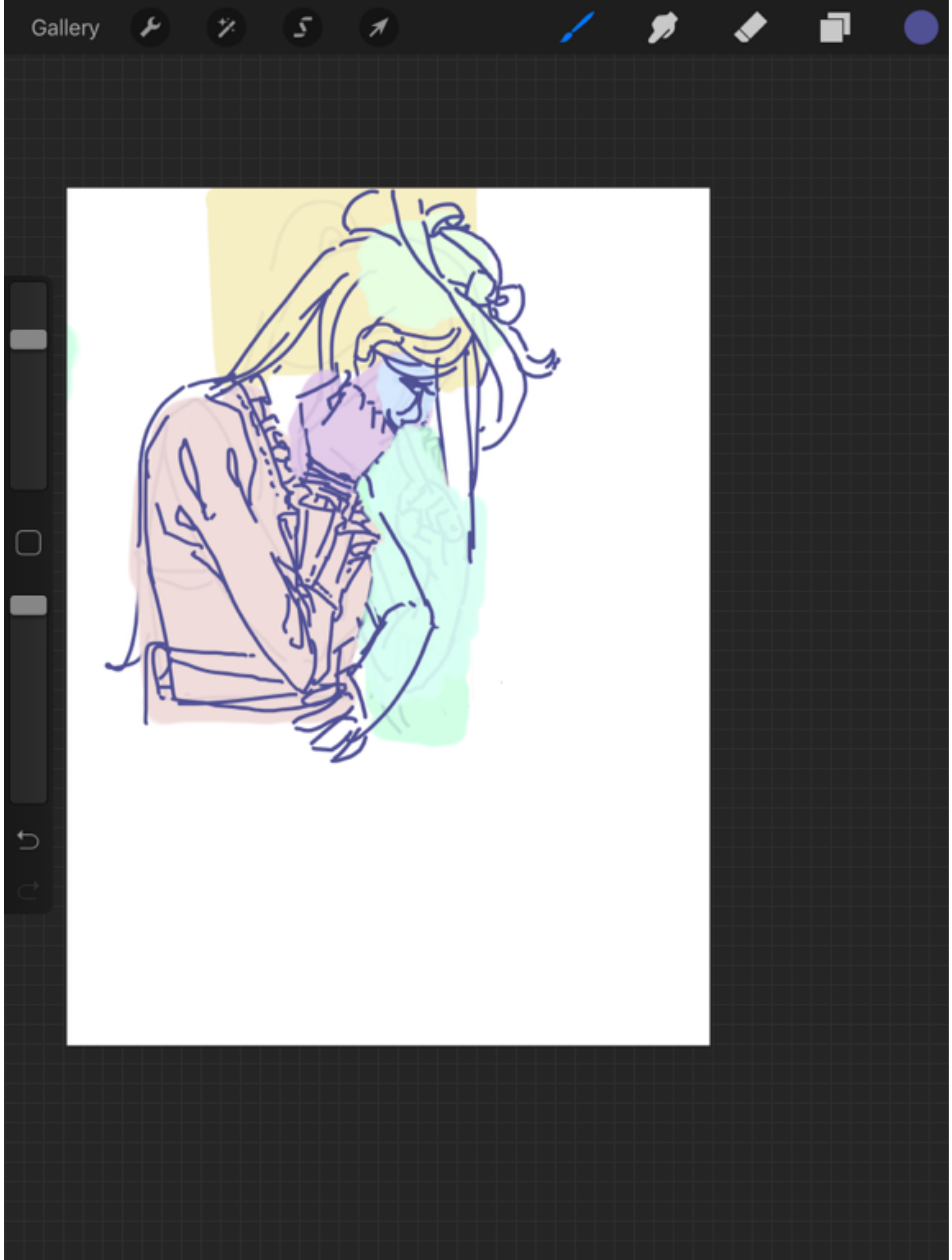

Gallery

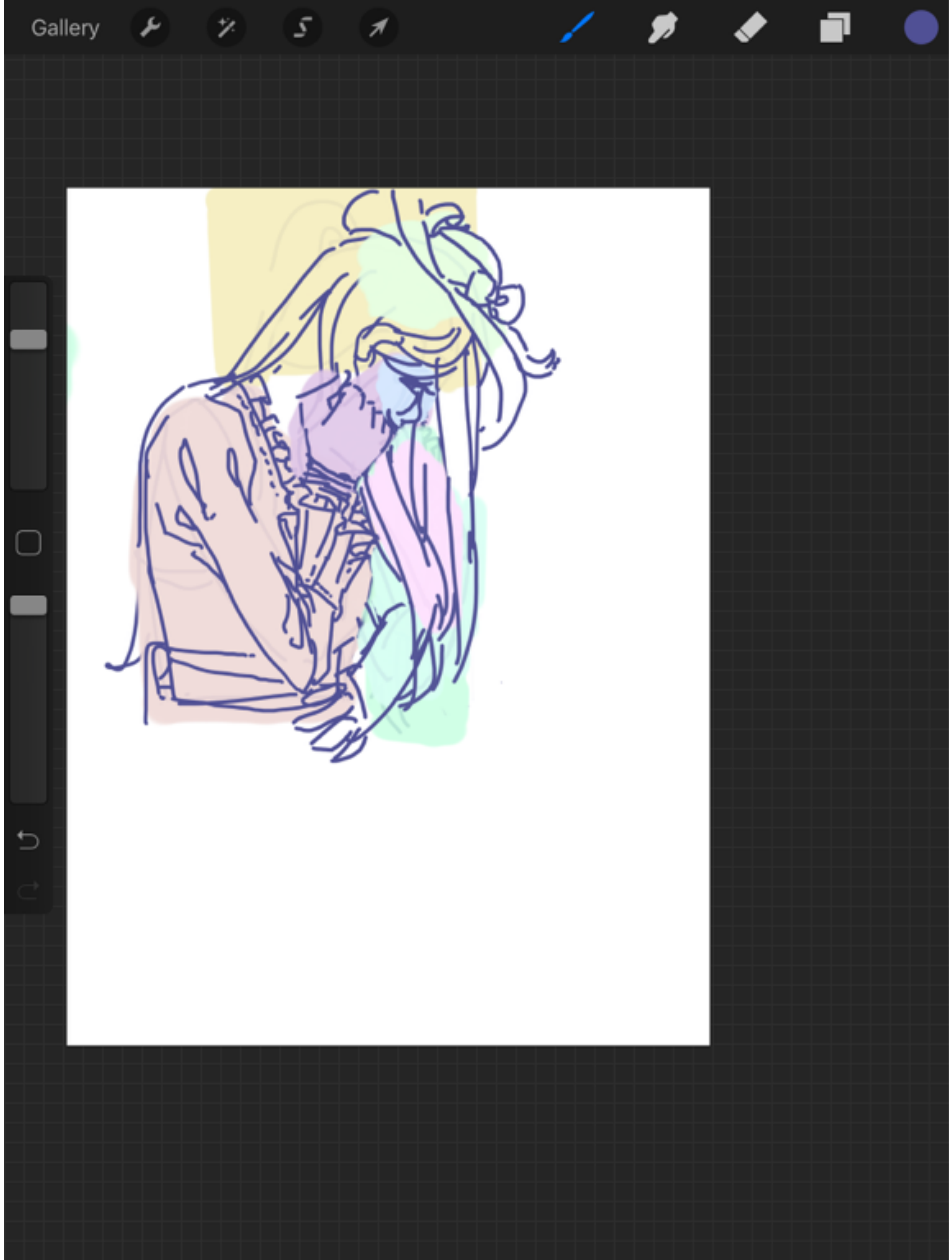

Gallery


Gallery

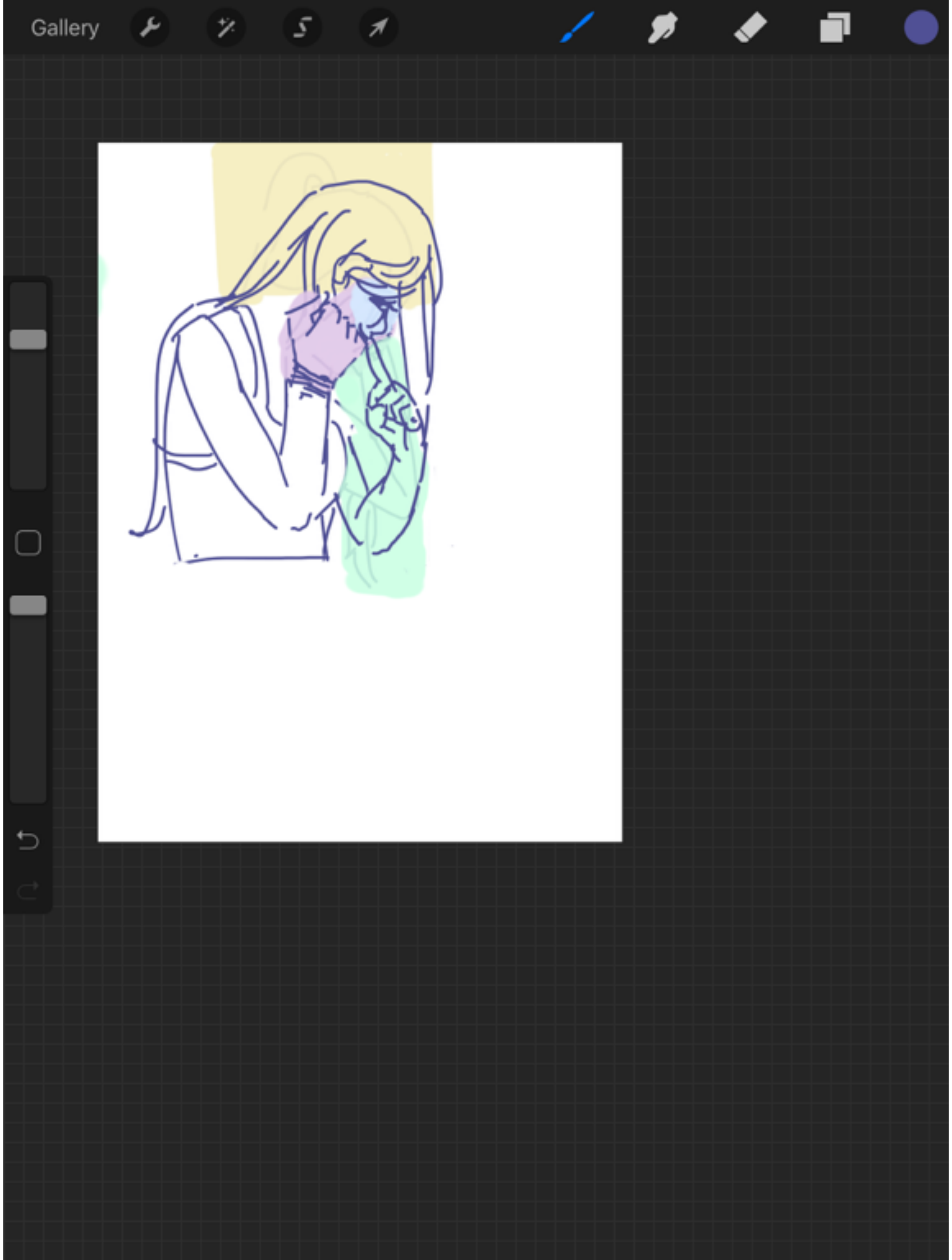

Gallery

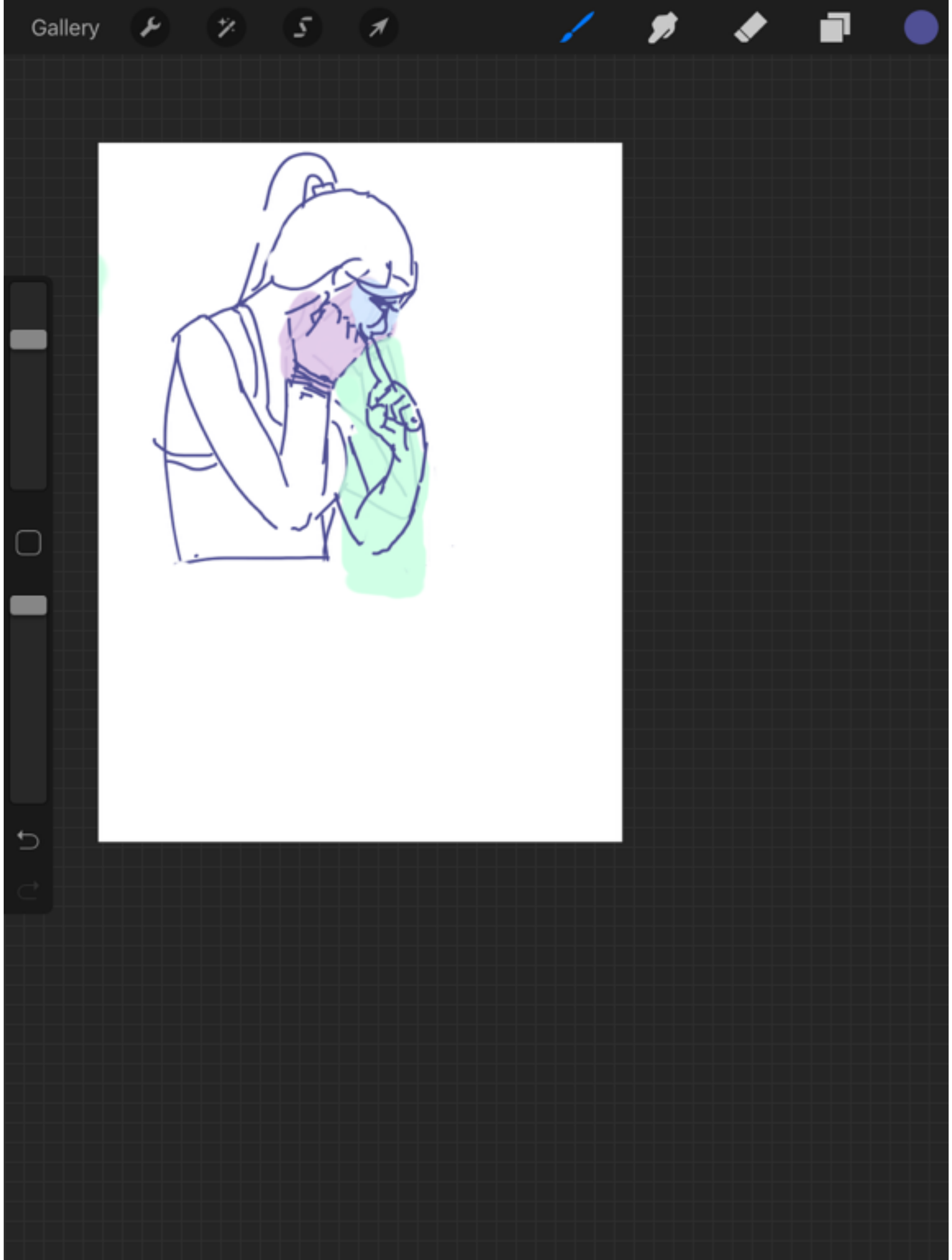

Gallery

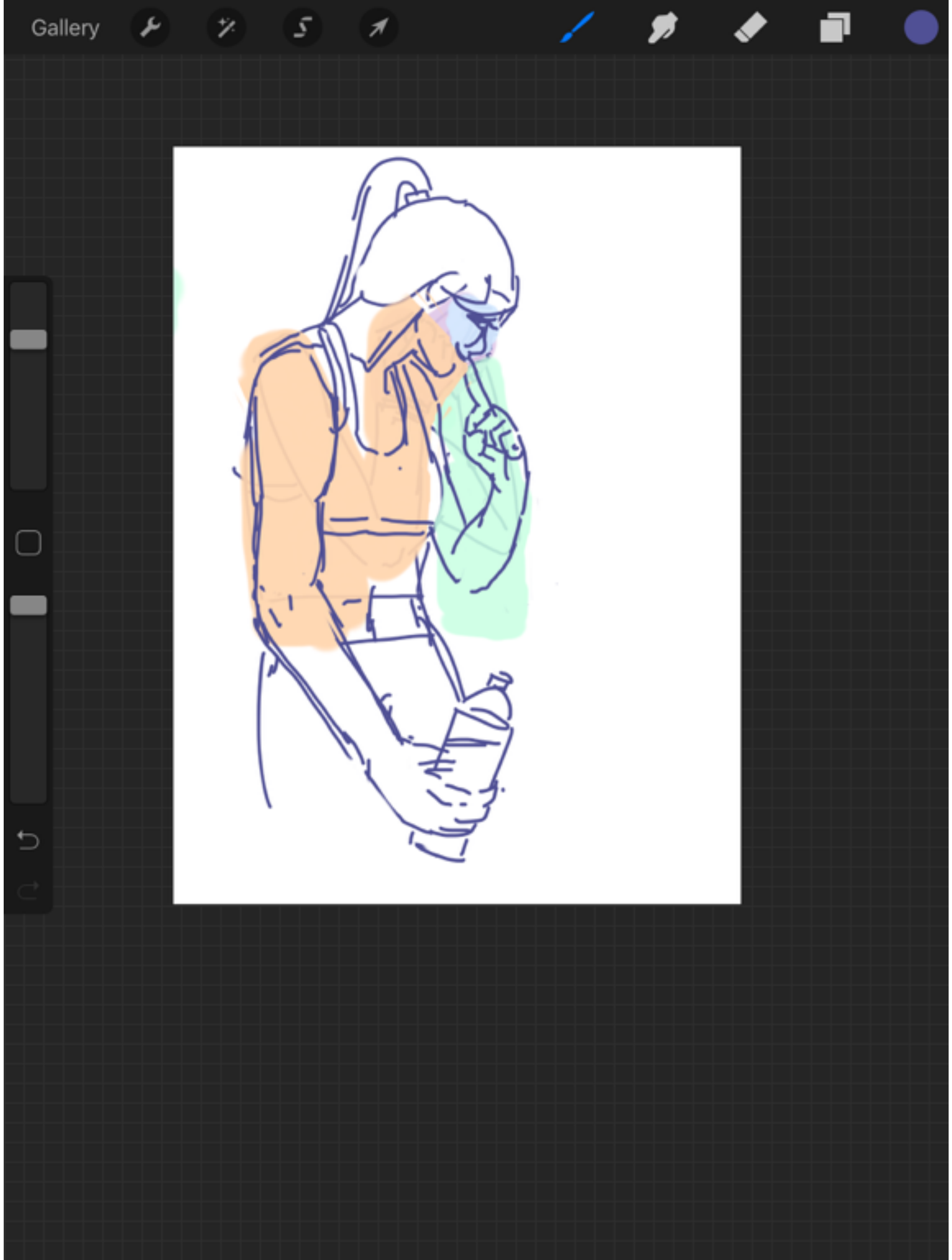

Gallery

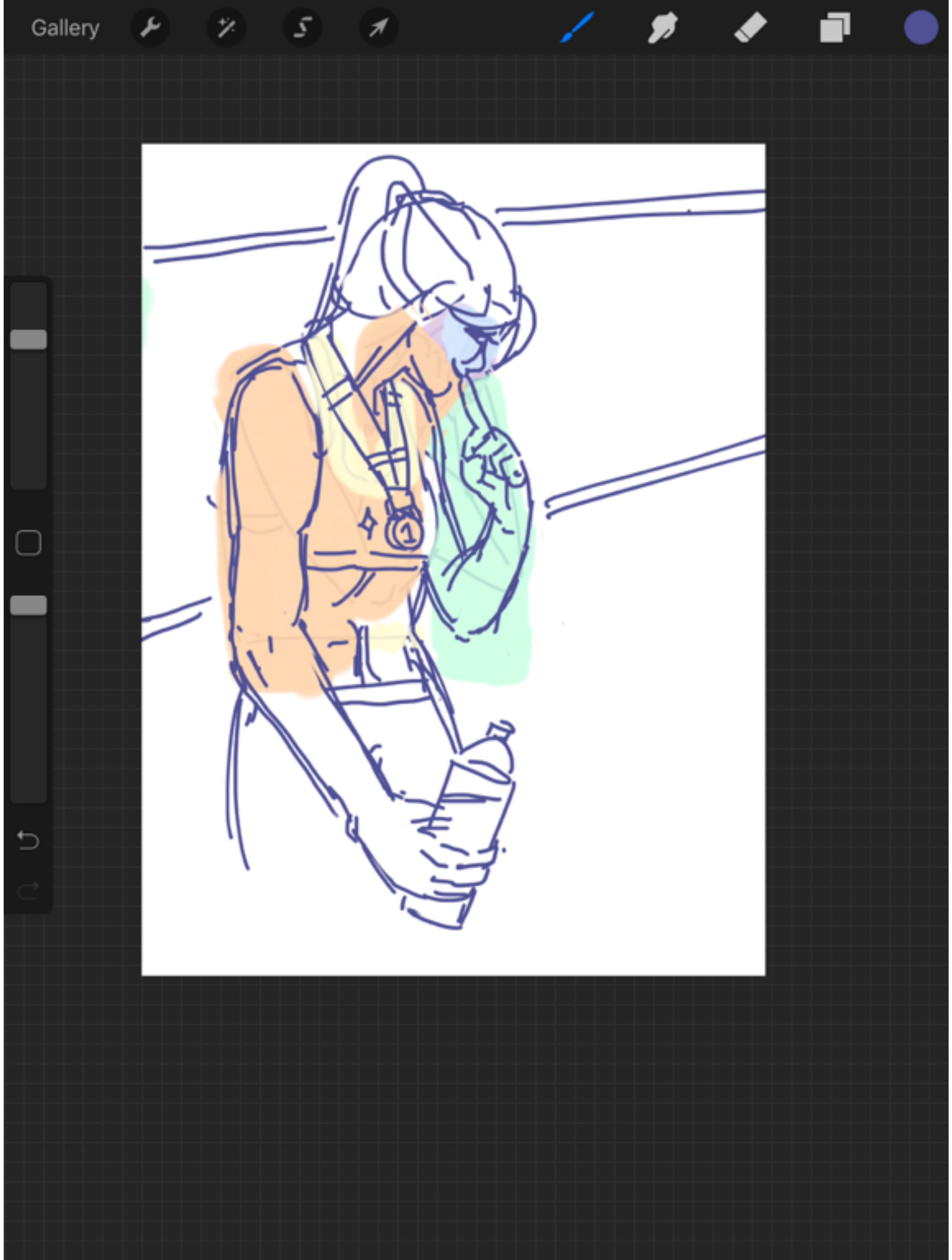

Gallery

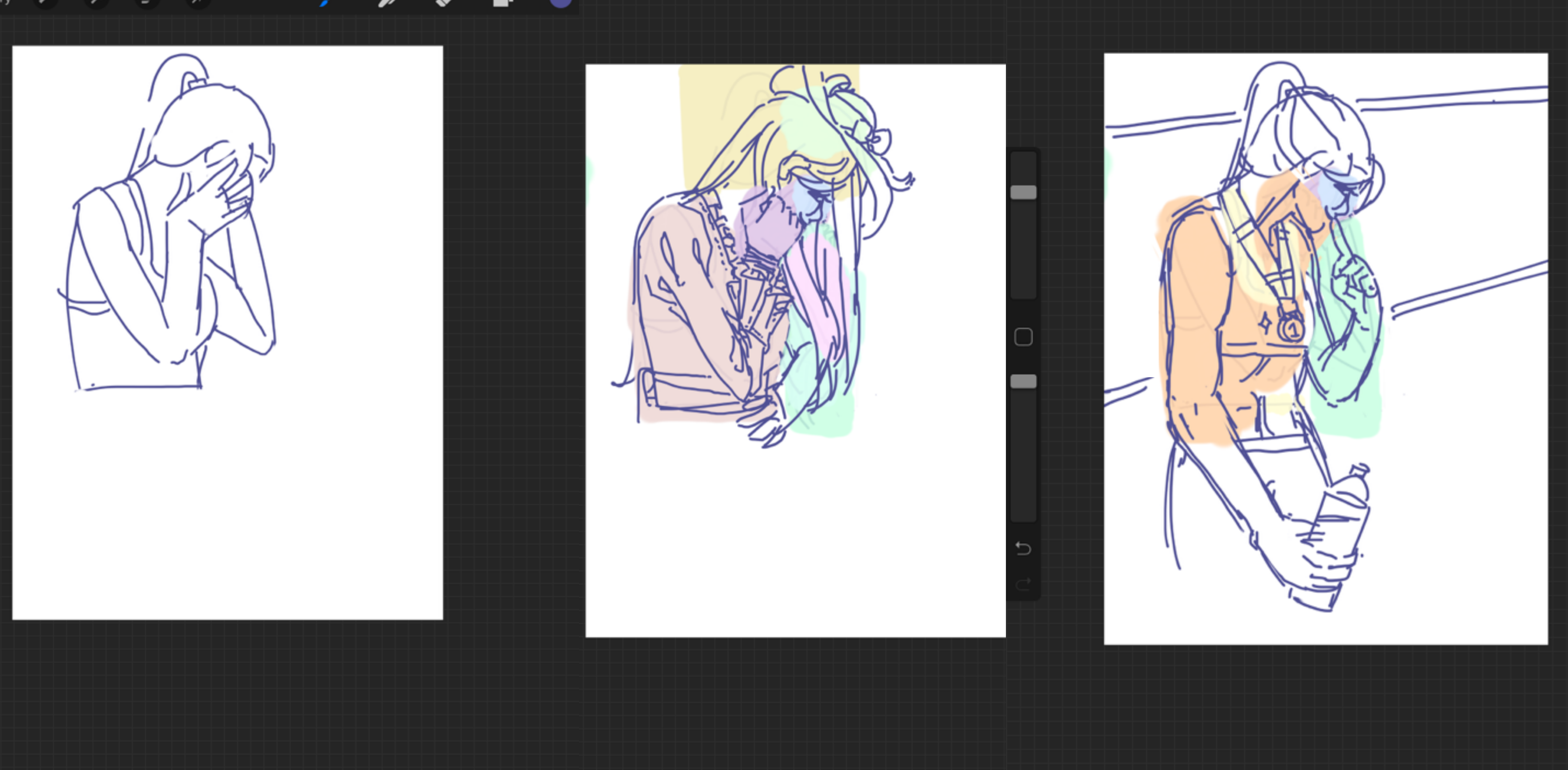
selective
backtracking




Gallery


Gallery


Gallery

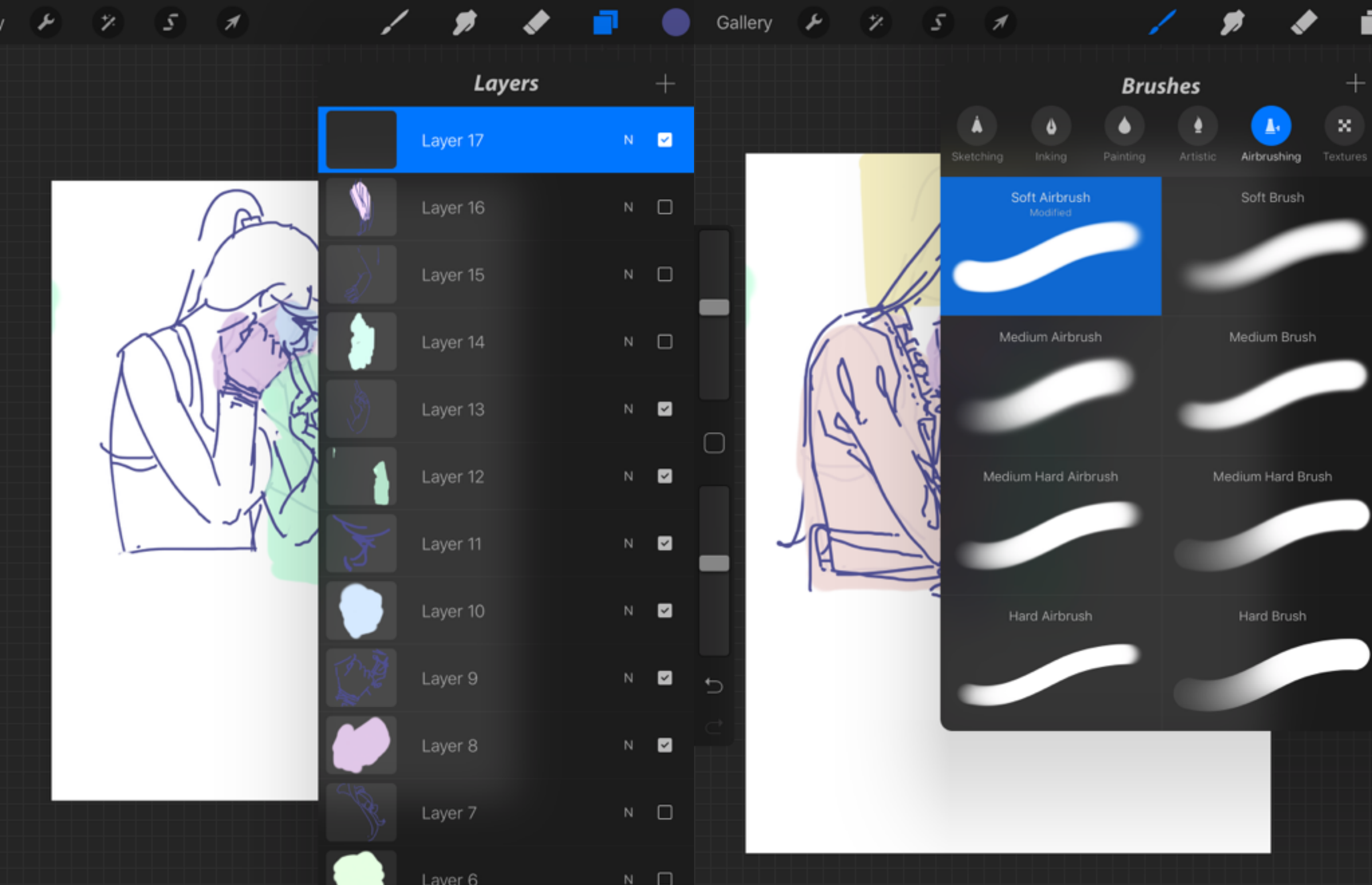


The act of writing inevitably changes what you want to say…

[It's] a process, not just of setting down a concept already fully formed in the mind, but of **discovering what it is you meant in the first place.**

First Draft of the Revolution
Emily Short & Liza Daly

Semantic spot process is richer than stroke-based process!
(video)



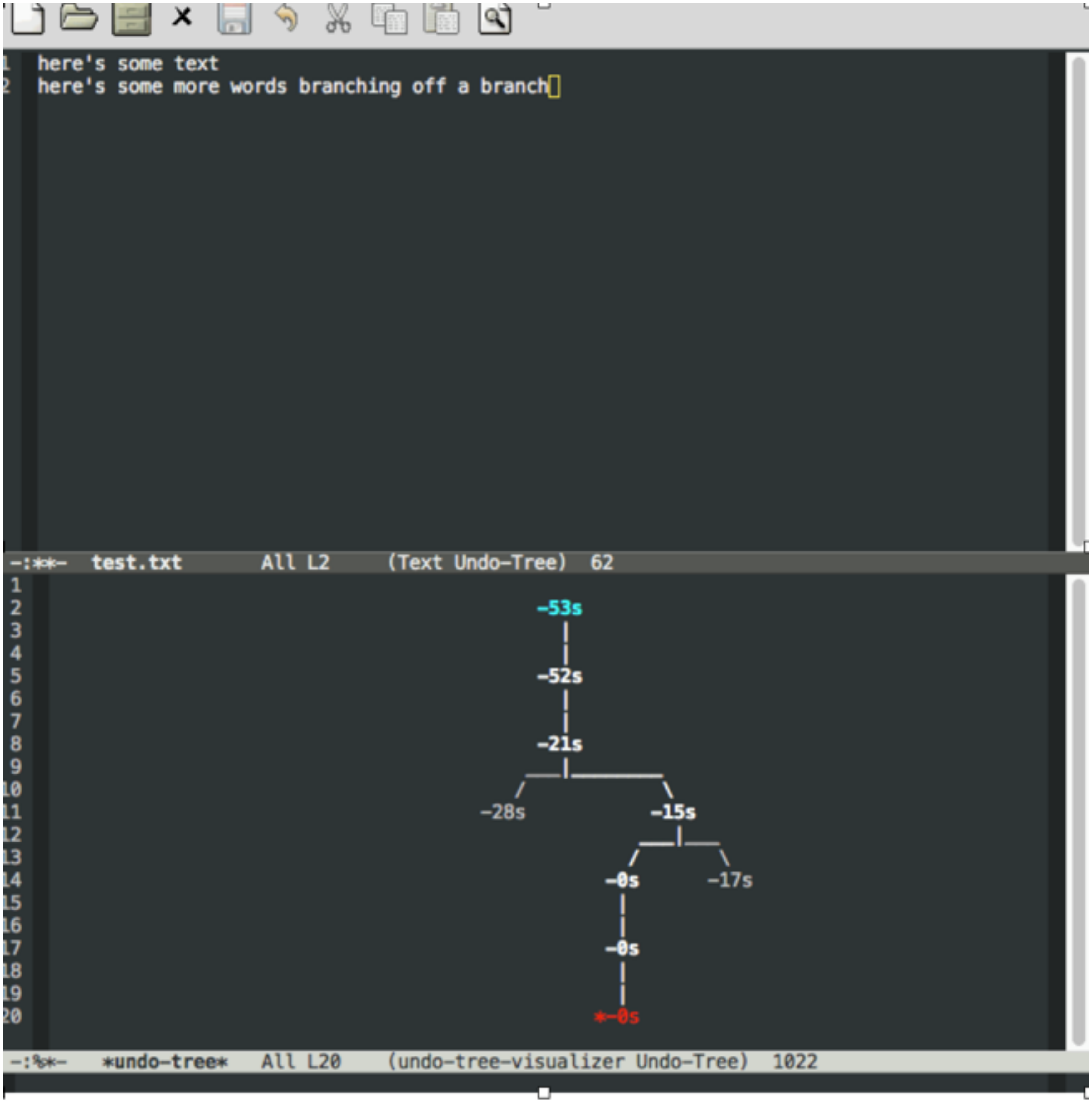
complicated to manually implement: let's build a tool

Related work

Rich manipulations on object and history

CAUSALITY: A Conceptual Model of Interaction History (Nancel and Cockburn)

video: undo trees in emacs (Toby Cubitt)



Case study for prose:
how can we turn desire paths into sidewalks?

Penrose objective function synthesis 4/3/17

File Edit View Insert Format Tools Table Add-ons Help Last edit was on May 15

katherine.ye@gmail.com

Comments Share

100% Normal text Arial 11 More

[[(Float, Setting)]]

e.g.

[[(xc, varying), (yc, varying), (r, fixed)], [(xl, varying), (yl, varying), (w, fixed), (h, fixed)], …] (it has to be a list of lists bc n-tuples are different types… unless you want a sum type)

split: when we flatten, keep the annotations, and just split using the annotations

SPLIT had better preserve the order too

can i cut out any of the intermediate steps?

so, what's automatic, and what needs to be specified? and what are the assumptions?

write each object unpack fn, maybe add annotations (but for now, add annotations when flattening that assume r, s are fixed)

```
-- make sure this matches circPack and labelPack
-- annotations are specified inline here. this is per type, not per value (i.e. all circles have the same fixed parameters). but you could generalize it to per-value by adding or overriding annotations globally after the unpacking
unpackObj :: Obj -> [(Float, Annotation)]
unpackObj (C c) = [(xc c, Varying), (yc c, Varying), (r c, Fixed)]
unpackObj (L l) = [(xl l, Varying), (yl l, Varying), (w l, Fixed), (h l, Fixed)]

-- split out because pack needs this annotated list of lists
unpackAnnotate :: [Obj] -> [ [(Float, Annotation)] ]
unpackAnnotate objs = map unpackObj objs

-- TODO check it preserves order
splitFV :: [ (Float, Annotation) ] -> (Fixed, Varying)
splitFV annotated = foldr chooseList ([], []) annotated
    where chooseList :: (Fixed, Varying) -> (Float, Annotation) -> (Fixed, Varying)
        chooseList (f, v) (x, Fix) = (x : f, v)
        chooseList (f, v) (x, Vary) = (f, x : v)

-- preserves the order of the objects' parameters
-- e.g. unpackSplit [Circ {xc varying, r fixed}, Label {xl varying, h fixed} ] = ( [r, h], [xc, xl] )
unpackSplit :: [Obj] -> (Fixed, Varying)
unpackSplit objs = let annotatedList = concat $ unpackAnnotate objs in
    splitFV annotatedL ist
```

Penrose objective function synthesis 4/3/17

katherine.ye@gmail.com

File Edit View Insert Format Tools Table Add-ons Help Last edit was on May 15

Comments Share

100% Normal text Arial 11 B I U A More

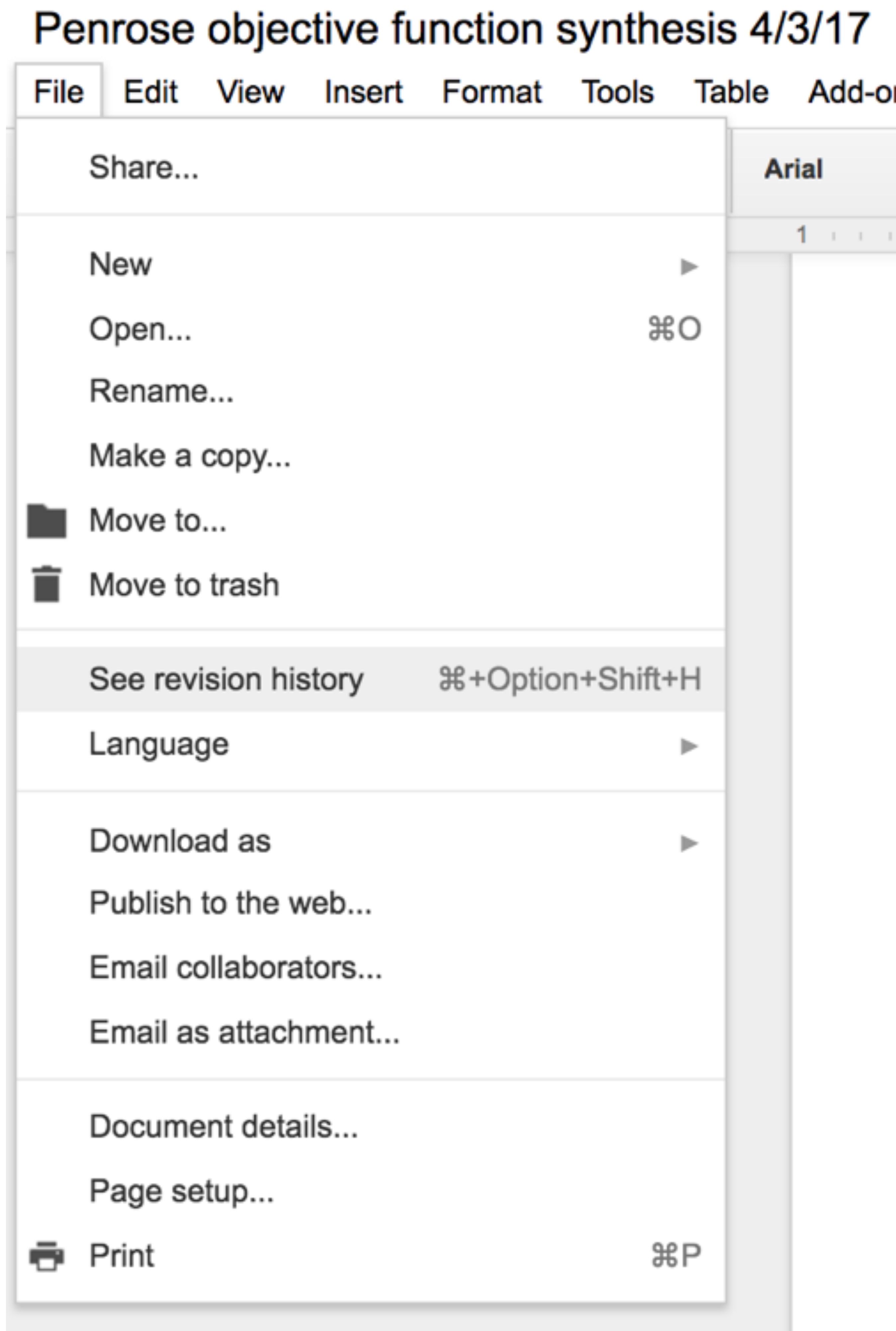
2 of 13

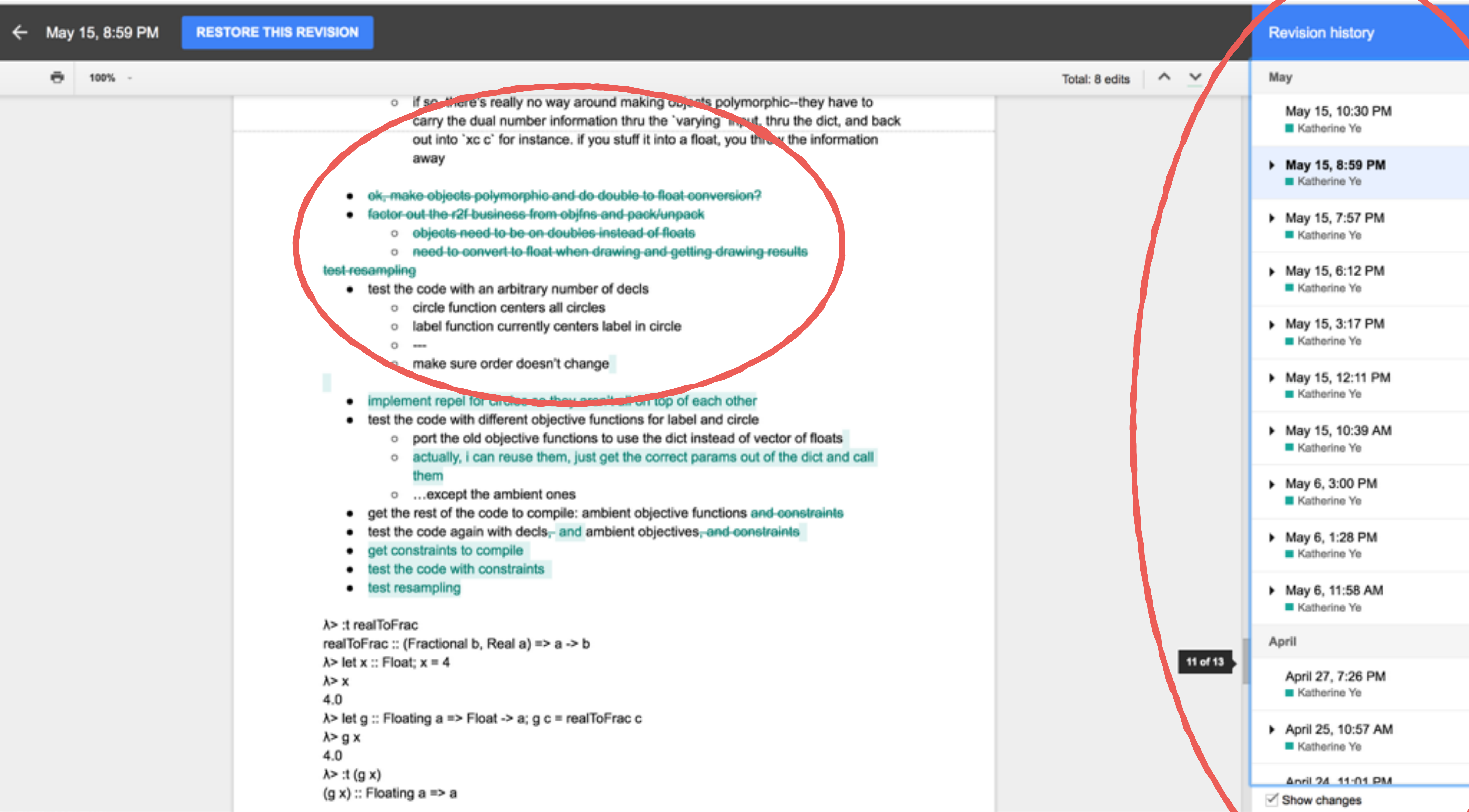
[[(Float, Setting)]]
e.g.
[[(xc, varying), (yc, varying), (r, fixed)], [(xl, varying), (yl, varying), (w, fixed), (h, fixed)], …] (it has to be a list of lists bc n-tuples are different types… unless you want a sum type)

split: when we flatten, keep the annotations, and just split using the annotations
SPLIT had better preserve the order too
can i cut out any of the intermediate steps?

so, what's automatic, and what needs to be specified? and what are the assumptions?
write each object unpack fn, maybe add annotations (but for now, add annotations when flattening that assume r, s are fixed)

```
-- make sure this matches circPack and labelPack
-- annotations are specified inline here. this is per type, not per value (i.e. all circles have the same fixed parameters). but you could generalize it to per-value by adding or overriding annotations globally after the unpacking
unpackObj :: Obj -> [(Float, Annotation)]
unpackObj (C c) = [(xc c, Varying), (yc c, Varying), (r c, Fixed)]
unpackObj (L l) = [(xl l, Varying), (yl l, Varying), (w l, Fixed), (h l, Fixed)]

-- split out because pack needs this annotated list of lists
unpackAnnotate :: [Obj] -> [ [(Float, Annotation)] ]
unpackAnnotate objs = map unpackObj objs

-- TODO check it preserves order
splitFV :: [ (Float, Annotation) ] -> (Fixed, Varying)
splitFV annotated = foldr chooseList ([], []) annotated
        where chooseList :: (Fixed, Varying) -> (Float, Annotation) -> (Fixed, Varying)
                chooseList (f, v) (x, Fix) = (x : f, v)
                chooseList (f, v) (x, Vary) = (f, x : v)

-- preserves the order of the objects' parameters
-- e.g. unpackSplit [Circ {xc varying, r fixed}, Label {xl varying, h fixed} ] = ( [r, h], [xc, xl] )
unpackSplit :: [Obj] -> (Fixed, Varying)
unpackSplit objs = let annotatedList = concat $ unpackAnnotate objs in
        splitFV annotatedList
```

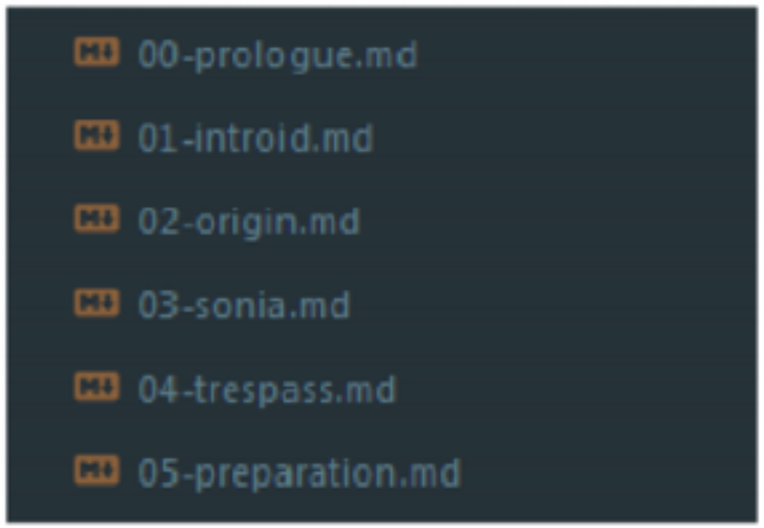

Penrose objective function synthesis 4/3/17
File
Edit
View
Insert
Format
Tools
Table
Add-or
Arial
Share...
New
Open... ⌘O
Rename...
Make a copy...
Move to...
Move to trash
See revision history ⌘+Option+Shift+H
Language
Download as
Publish to the web...
Email collaborators...
Email as attachment...
Document details...
Page setup...
Print ⌘P



no user-set checkpoints or comments

I am currently writing an as-of-unnamed science fantasy novel using Sublime Text 3 and Markdown.

Every chapter is a file...



....and the whole manuscript is put together using a bash script that concatenates the chapters in name order, and then outputs a .docx file.

The reason why I'm using this setup is to make sure that all of the source text is in plaintext, primarily so that it can work with version control. The entire novel is backed by a Mercurial repository. This generally works pretty well.

(it's reassuring to have backups to reference!)

Diana Liao (highlights added)

Commits themselves aren't terribly useful for diffing for the fact that they are based on lines, which for a novel ends up being entire paragraphs. (Also, my commits are just a big mess.) I don't use them to compare differences, just to have a backup of prose that I wrote and might've cut, in case I want to revisit it.

In which I edit too many chapters in one sitting:

Files changed (15)

+4	-1	M	03-sonia.md
+3	-0	M	09-investigation.md
+2	-31	M	14n2-hideout.md
+44	-0	M	16x-town.md
+103	-2	M	21-awakening.md
+7	-43	M	21x-inevitability.md
+116	-5	M	22-celebratory.md
+117	-0	A	22x-initial.md
+3	-1	M	24-lone.md
+0	-0	M	build/all.docx
+789	-111	M	build/all.md

Diana Liao (highlights added)

(don't look at the rest of this talk if you're a CCS reviewer!)

```
-In this work we focus on proving the pseudorandomness property of HMAC-DRBG. As with traditional paper proofs, our approach is to
 show the computational indistinguishability of two main ``experiments.'' In the first (or ``real'') experiment, the adversary
 interacts with an oracle that  produces generator formulated using the HMAC-DRBG protocol. In the second (or ``ideal'')
 experiment, the adversary interacts with an oracle that produces uniformly random output. Our goal is to connect these experiments
 via a series of intermediate ``hybrid'' experiments, which we will use to show that the real and ideal experiments are
 computationally indistinguishable from the point of any probabilistic polynomial time (p.p.t.) adversary.
+We prove the pseudorandomness property of HMAC-DRBG. As with traditional proofs, we show the computational indistinguishability of
 two main ``experiments.'' In the first (or ``real'') experiment, the adversary interacts with an oracle that  produces generator
 formulated using the HMAC-DRBG protocol. In the second (or ``ideal'') experiment, the adversary interacts with an oracle that
 produces uniformly random output. Our goal is to connect these experiments via a series of intermediate ``hybrid'' experiments,
 which we will use to show that the real and ideal experiments are computationally indistinguishable from the point of any
 probabilistic polynomial time (p.p.t.) adversary.
```

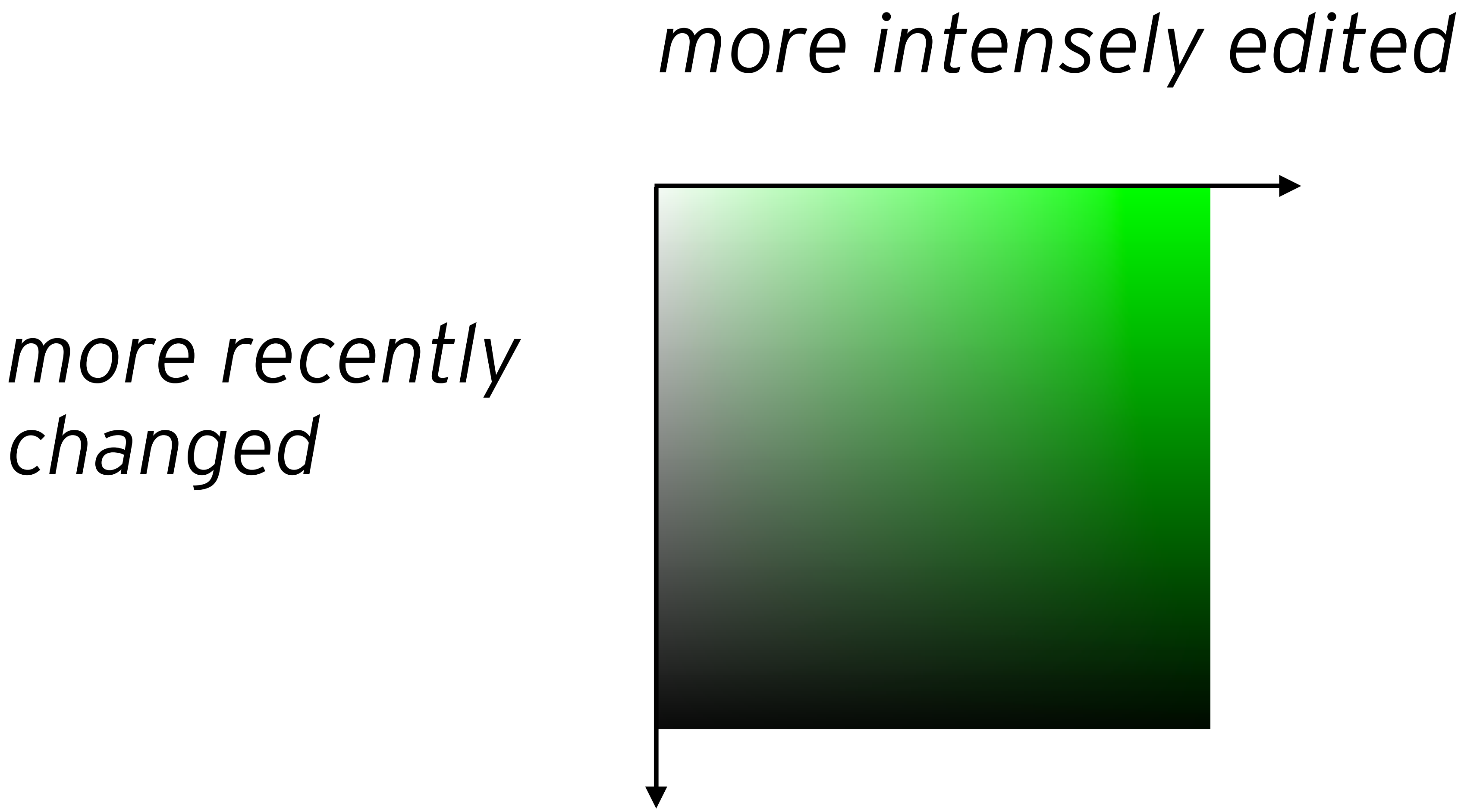
In prose, how can we bridge the gap between history and version control?

How can we create a skimmable history?

How did this text change over time?

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

If an adversary compromises a PRG, the results could be catastrophic: the adversary could, for example, predict private keys that will be generated as part of the protocol. PRG compromise is both easy for adversaries and difficult to prevent. This is because PRGs are difficult to verify; the state of the art is statistical tests.


more intensely edited
more recently changed

PRGs are important because all cryptosystems depend on them.

PRGs are important because all cryptosystems depend on them. If you break a PRG, it breaks the whole cryptosystem.

PRGs are important because all cryptosystems depend on them. If you break a PRG, it breaks the whole cryptosystem.

Intuitively, a PRG

PRGs are important because all cryptosystems depend on them. If you break a PRG, it breaks the whole cryptosystem.

Almost all cryptosystems require a large amount of randomness to initialize. But often we only have a small amount of true randomness. Our solution

PRGs are important because all cryptosystems depend on them. If you break a PRG, it breaks the whole cryptosystem.

Almost all cryptosystems require a large amount of randomness to initialize things like nonces, keys, and initialization vectors. But often we only have a small amount of true randomness. PRGs are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

Almost all cryptosystems require a large amount of randomness to initialize things like nonces, keys, and initialization vectors. But often we only have a small amount of true randomness. PRGs are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

PRGs are important because all cryptosystems depend on them. If you break a PRG, it breaks the whole cryptosystem.

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. PRGs are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

If an adversary compromises a PRG, the adversary

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

If an adversary compromises a PRG, the adversary can often completely decrypt communications that depend on the PRG without being detected. This is because PRGs are difficult to verify; the state of the art is statistical tests.

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

If an adversary compromises a PRG, the adversary could, for example, predict private keys. PRG compromise is both easy for adversaries and difficult to prevent. This is because PRGs are difficult to verify; the state of the art is statistical tests.

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

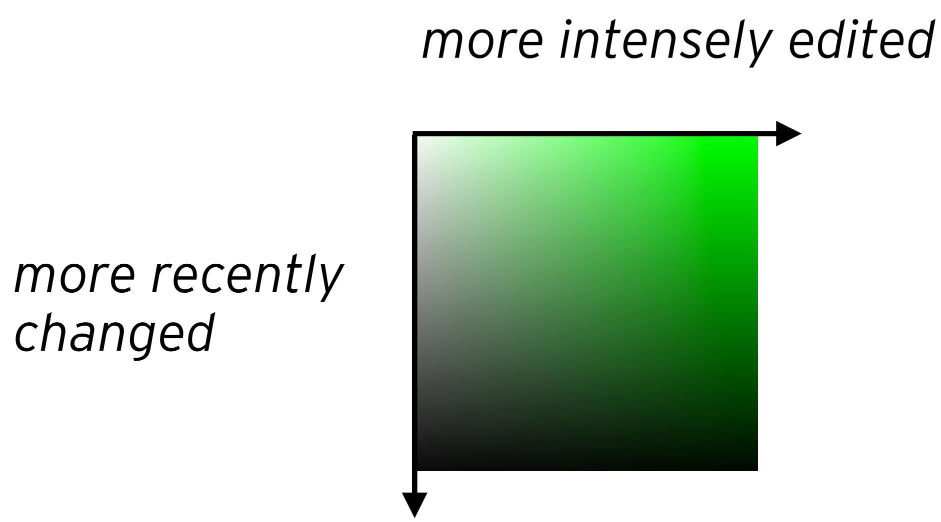
If an adversary compromises a PRG, the results could be catastrophic: the adversary could, for example, predict private keys that will be generated as part of the protocol. PRG compromise is both easy for adversaries and difficult to prevent. This is because PRGs are difficult to verify; the state of the art is statistical tests.

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

If an adversary compromises a PRG, the results could be catastrophic: the adversary could, for example, predict private keys that will be generated as part of the protocol. PRG compromise is both easy for adversaries and difficult to prevent. This is because PRGs are difficult to verify; the state of the art is statistical tests.

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

If an adversary compromises a PRG, the results could be catastrophic: the adversary could, for example, predict private keys that will be generated as part of the protocol. PRG compromise is both easy for adversaries and difficult to prevent. This is because PRGs are difficult to verify; the state of the art is statistical tests.



Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

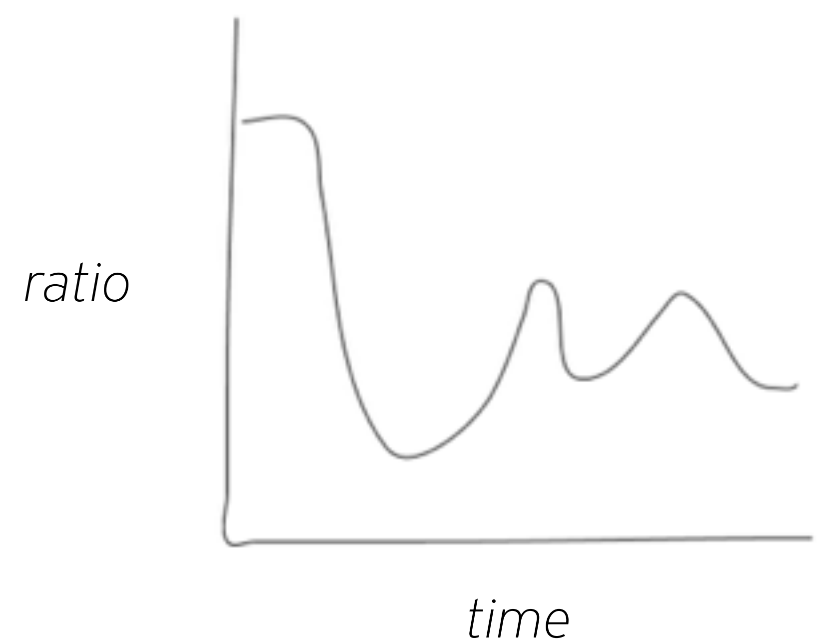
If an adversary compromises a PRG, the results could be catastrophic: the adversary could, for example, predict private keys that will be generated as part of the protocol. PRG compromise is both easy for adversaries and difficult to prevent. This is because PRGs are difficult to verify; the state of the art is statistical tests.

Almost all cryptosystems require a large amount of randomness to initialize elements of the state such as nonces, keys, and initialization vectors. But true randomness is typically available only in small amounts. Pseudo-random generators, or PRGs, are used to “stretch” this small amount of true randomness into a large amount of randomness that is computationally indistinguishable from true randomness.

If an adversary compromises a PRG, the results could be catastrophic: the adversary could, for example, predict private keys that will be generated as part of the protocol. PRG compromise is both easy for adversaries and difficult to prevent. This is because PRGs are difficult to verify; the state of the art is statistical tests.

undo/delete-to-content ratio

$$= 34 / 144$$
$$= 0.24$$



ABSTRACT

We have formalized the functional specification of HMAC-DRBG (NIST 800-90A), and we have proved its cryptographic security—that its output is pseudorandom—using a hybrid game-based proof. We have also proved that the mbedTLS implementation (C program) correctly implements this functional specification. That proof composes with an existing C compiler correctness proof to guarantee, end-to-end, that the machine language program gives strong pseudorandomness. All proofs (hybrid games, C program verification, compiler, and their composition) are machine-checked in the Coq proof assistant. Our proofs are modular: the hybrid game proof holds on any implementation of HMAC-DRBG that satisfies our functional specification. Therefore, our functional specification can serve as a high-assurance reference.

1 INTRODUCTION

Cryptographic systems require large amounts of randomness to generate keys, nonces and initialization vectors. Because many computers lack the large amount of high-quality physical randomness needed to generate these values, most cryptographic devices rely on pseudorandom generators (also known as *deterministic random bit generators* or DRBGs) to "stretch" small amounts of true randomness into large amounts of pseudorandom output.[1]

Pseudorandom generators are crucial to security. Compromising a generator (for example, by selecting malicious algorithm constants, or exfiltrating generator state) can harm the security of nearly any cryptosystem built on top of it. The harm can be catastrophic: for example, an adversary who can predict future outputs of the generator may be able to predict private keys that will be generated, or recover long term keys used as input to the protocol execution. Moreover, it can be practically impossible to test that a DRBG has been compromised if one is limited to black-box testing: current validation standards [21, 26, 28] primarily resort to statistical tests and test vectors, neither of which guarantee that the output is pseudorandom. Yet, despite the importance of DRBGs, their development has not received the scrutiny it deserves.

Many catastrophic flaws in random number generators have been uncovered at both the design level and the implementation level. Some bugs arise from simple programming mistakes. The Debian DRBG, though open-source, was broken for two years because a line of code was erroneously removed, weakening the DRBG's seeding process [25]. Several recent projects have identified and factored large numbers of weak RSA keys produced due to improper generator seeding [9, 18, 24]. A bug in entropy use in the Android DRBG resulted in the theft of $5,700 of Bitcoin [17].

While those flaws were accidental, some are malicious. Adversaries intentionally target DRBGs because that's an easy way to break the larger cryptosystem. The most notorious example is the NSA's alleged backdooring of the Dual EC DRBG standard [4, 29, 32]. In the Dual EC design, the malicious choice of a single algorithm parameter (an elliptic curve point Q) allows for full state recovery of the DRBG given only a small amount of raw generator output. This enables the passive decryption of protocols such as TLS and IPSEC/IKE [11, 12, 32]. From 2012 to 2015 this backdoor was exploited by unauthorized parties to insert a passive decryption backdoor into Juniper NetScreen VPN devices [11, 35]. Remarkably, a related DRBG flaw in Fortinet VPNs created a similar vulnerability in those devices during the same time period [20].

A key weakness in the deployment of DRBGs is that current government standards both encourage specific designs and lack rigor. The FIPS validation process (required by the U.S. government for certain types of cryptographic device) mandates the use of an "approved" NIST DRBG in every cryptographic module. As a consequence, a small number of DRBG designs have become ubiquitous throughout the industry. These algorithms lack formal security proofs, and their design processes were neither open nor rigorous—in fact, the designs were found to include errors. Even worse, it is easy to implement certain of these DRBGs such that the output of the generator is predictable (given detailed knowledge of the implementation), and yet without this knowledge *the output of the generator is computationally indistinguishable from random.* Unfortunately, the existing formal validation processes for verifying the correctness of DRBG implementations are weak and routinely ignore entire classes of flaws. Given that the widespread deployment of a weak DRBG can undermine the security of entire computer networks, we need a better way to validate these critical systems.

1.1 Contributions

DRBGs have the special property that testing, even sophisticated statistical fuzz-testing, cannot assure the security of an implementation. Even at the level of a pure cryptographic protocol, proving the security of a DRBG construction can be quite challenging. (Hybrid-game-based proofs are the only known technique, but for nontrivial DRBGs those proofs have so many technical steps that it is hard to trust them.) Therefore a new paradigm is required, in which the functional model of a DRBG is proved *with a machine-checked proof* to satisfy the appropriate PRF properties, and the C-language implementation (*and* its compilation to machine language) is proved

[1] A note on terminology: we use "entropy" loosely to denote randomness that is not predictable by an adversary. We use "sampled uniformly at random" and "ideally random" interchangeably. We use PRG, the acronym for "pseudo-random generator," to refer to the abstract cryptographic concept, whereas we use DRBG, the acronym for "deterministic random bit generator," to denote the specifications and implementations of PRGs. Instead of DRBG, some papers use "PRNG," the acronym for "pseudo-random number generator." The terms are synonymous.

ABSTRACT

We have formalized the functional specification of HMAC-DRBG (NIST 800-90A), and we have proved its cryptographic security—that its output is pseudorandom—using a hybrid game-based proof. We have also proved that the mbedTLS implementation (C program) correctly implements this functional specification. That proof composes with an existing C compiler correctness proof to guarantee, end-to-end, that the machine language program gives strong pseudorandomness. All proofs (hybrid games, C program verification, compiler, and their composition) are machine-checked in the Coq proof assistant. Our proofs are modular: the hybrid game proof holds on any implementation of HMAC-DRBG that satisfies our functional specification. Therefore, our functional specification can serve as a high-assurance reference.

1 INTRODUCTION

Cryptographic systems require large amounts of randomness to generate keys, nonces and initialization vectors. Because many computers lack the large amount of high-quality physical randomness needed to generate these values, most cryptographic devices rely on pseudorandom generators (also known as *deterministic random bit generators* or DRBGs) to "stretch" small amounts of true randomness into large amounts of pseudorandom output.[1]

Pseudorandom generators are crucial to security. Compromising a generator (for example, by selecting malicious algorithm constants, or exfiltrating generator state) can harm the security of nearly any cryptosystem built on top of it. The harm can be catastrophic: for example, an adversary who can predict future outputs of the generator may be able to predict private keys that will be generated, or recover long term keys used as input to the protocol execution. Moreover, it can be practically impossible to test that a DRBG has been compromised if one is limited to black-box testing: current validation standards [21, 26, 28] primarily resort to statistical tests and test vectors, neither of which guarantee that the output is pseudorandom. Yet, despite the importance of DRBGs, their development has not received the scrutiny it deserves.

Many catastrophic flaws in random number generators have been uncovered at both the design level and the implementation level. Some bugs arise from simple programming mistakes. The Debian DRBG, though open-source, was broken for two years because a line of code was erroneously removed, weakening the DRBG's seeding process [25]. Several recent projects have identified and factored large numbers of weak RSA keys produced due to improper generator seeding [9, 18, 24]. A bug in entropy use in the Android DRBG resulted in the theft of $5,700 of Bitcoin [17].

While those flaws were accidental, some are malicious. Adversaries intentionally target DRBGs because that's an easy way to break the larger cryptosystem. The most notorious example is the NSA's alleged backdooring of the Dual EC DRBG standard [4, 29, 32]. In the Dual EC design, the malicious choice of a single algorithm parameter (an elliptic curve point Q) allows for full state recovery of the DRBG given only a small amount of raw generator output. This enables the passive decryption of protocols such as TLS and IPSEC/IKE [11, 12, 32]. From 2012 to 2015 this backdoor was exploited by unauthorized parties to insert a passive decryption backdoor into Juniper NetScreen VPN devices [11, 35]. Remarkably, a related DRBG flaw in Fortinet VPNs created a similar vulnerability in those devices during the same time period [20].

A key weakness in the deployment of DRBGs is that current government standards both encourage specific designs and lack rigor. The FIPS validation process (required by the U.S. government for certain types of cryptographic device) mandates the use of an "approved" NIST DRBG in every cryptographic module. As a consequence, a small number of DRBG designs have become ubiquitous throughout the industry. These algorithms lack formal security proofs, and their design processes were neither open nor rigorous—in fact, the designs were found to include errors. Even worse, it is easy to implement certain of these DRBGs such that the output of the generator is predictable (given detailed knowledge of the implementation), and yet without this knowledge *the output of the generator is computationally indistinguishable from random*. Unfortunately, the existing formal validation processes for verifying the correctness of DRBG implementations are weak and routinely ignore entire classes of flaws. Given that the widespread deployment of a weak DRBG can undermine the security of entire computer networks, we need a better way to validate these critical systems.

1.1 Contributions

DRBGs have the special property that testing, even sophisticated statistical fuzz-testing, cannot assure the security of an implementation. Even at the level of a pure cryptographic protocol, proving the security of a DRBG construction can be quite challenging. (Hybrid-game-based proofs are the only known technique, but for nontrivial DRBGs those proofs have so many technical steps that it is hard to trust them.) Therefore a new paradigm is required, in which the functional model of a DRBG is proved *with a machine-checked proof* to satisfy the appropriate PRF properties, and the C-language implementation (*and* its compilation to machine language) is proved

[1] A note on terminology: we use "entropy" loosely to denote randomness that is not predictable by an adversary. We use "sampled uniformly at random" and "ideally random" interchangeably. We use PRG, the acronym for "pseudo-random generator," to refer to the abstract cryptographic concept, whereas we use DRBG, the acronym for "deterministic random bit generator," to denote the specifications and implementations of PRGs. Instead of DRBG, some papers use "PRNG," the acronym for "pseudo-random number generator." The terms are synonymous.

ABSTRACT

We have formalized the functional specification of HMAC-DRBG (NIST 800-90A), and we have proved its cryptographic security—that its output is pseudorandom—using a hybrid game-based proof. We have also proved that the mbedTLS implementation (C program) correctly implements this functional specification. That proof composes with an existing C compiler correctness proof to guarantee, end-to-end, that the machine language program gives strong pseudorandomness. All proofs (hybrid games, C program verification, compiler, and their composition) are machine-checked in the Coq proof assistant. Our proofs are modular: the hybrid game proof holds on any implementation of HMAC-DRBG that satisfies our functional specification. Therefore, our functional specification can serve as a high-assurance reference.

1 INTRODUCTION

Cryptographic systems require large amounts of randomness to generate keys, nonces and initialization vectors. Because many computers lack the large amount of high-quality physical randomness needed to generate these values, most cryptographic devices rely on pseudorandom generators (also known as *deterministic random bit generators* or DRBGs) to "stretch" small amounts of true randomness into large amounts of pseudorandom output.[1]

Pseudorandom generators are crucial to security. Compromising a generator (for example, by selecting malicious algorithm constants, or exfiltrating generator state) can harm the security of nearly any cryptosystem built on top of it. The harm can be catastrophic: for example, an adversary who can predict future outputs of the generator may be able to predict private keys that will be generated, or recover long term keys used as input to the protocol execution. Moreover, it can be practically impossible to test that a DRBG has been compromised if one is limited to black-box testing: current validation standards [21, 26, 28] primarily resort to statistical tests and test vectors, neither of which guarantee that the output is pseudorandom. Yet, despite the importance of DRBGs, their development has not received the scrutiny it deserves.

Many catastrophic flaws in random number generators have been uncovered at both the design level and the implementation level. Some bugs arise from simple programming mistakes. The Debian DRBG, though open-source, was broken for two years because a line of code was erroneously removed, weakening the DRBG's seeding process [25]. Several recent projects have identified and factored large numbers of weak RSA keys produced due to improper generator seeding [9, 18, 24]. A bug in entropy use in the Android DRBG resulted in the theft of $5,700 of Bitcoin [17].

While those flaws were accidental, some are malicious. Adversaries intentionally target DRBGs because that's an easy way to break the larger cryptosystem. The most notorious example is the NSA's alleged backdooring of the Dual EC DRBG standard [4, 29, 32]. In the Dual EC design, the malicious choice of a single algorithm parameter (an elliptic curve point Q) allows for full state recovery of the DRBG given only a small amount of raw generator output. This enables the passive decryption of protocols such as TLS and IPSEC/IKE [11, 12, 32]. From 2012 to 2015 this backdoor was exploited by unauthorized parties to insert a passive decryption backdoor into Juniper NetScreen VPN devices [11, 35]. Remarkably, a related DRBG flaw in Fortinet VPNs created a similar vulnerability in those devices during the same time period [20].

A key weakness in the deployment of DRBGs is that current government standards both encourage specific designs and lack rigor. The FIPS validation process (required by the U.S. government for certain types of cryptographic device) mandates the use of an "approved" NIST DRBG in every cryptographic module. As a consequence, a small number of DRBG designs have become ubiquitous throughout the industry. These algorithms lack formal security proofs, and their design processes were neither open nor rigorous—in fact, the designs were found to include errors. Even worse, it is easy to implement certain of these DRBGs such that the output of the generator is predictable (given detailed knowledge of the implementation), and yet without this knowledge *the output of the generator is computationally indistinguishable from random*. Unfortunately, the existing formal validation processes for verifying the correctness of DRBG implementations are weak and routinely ignore entire classes of flaws. Given that the widespread deployment of a weak DRBG can undermine the security of entire computer networks, we need a better way to validate these critical systems.

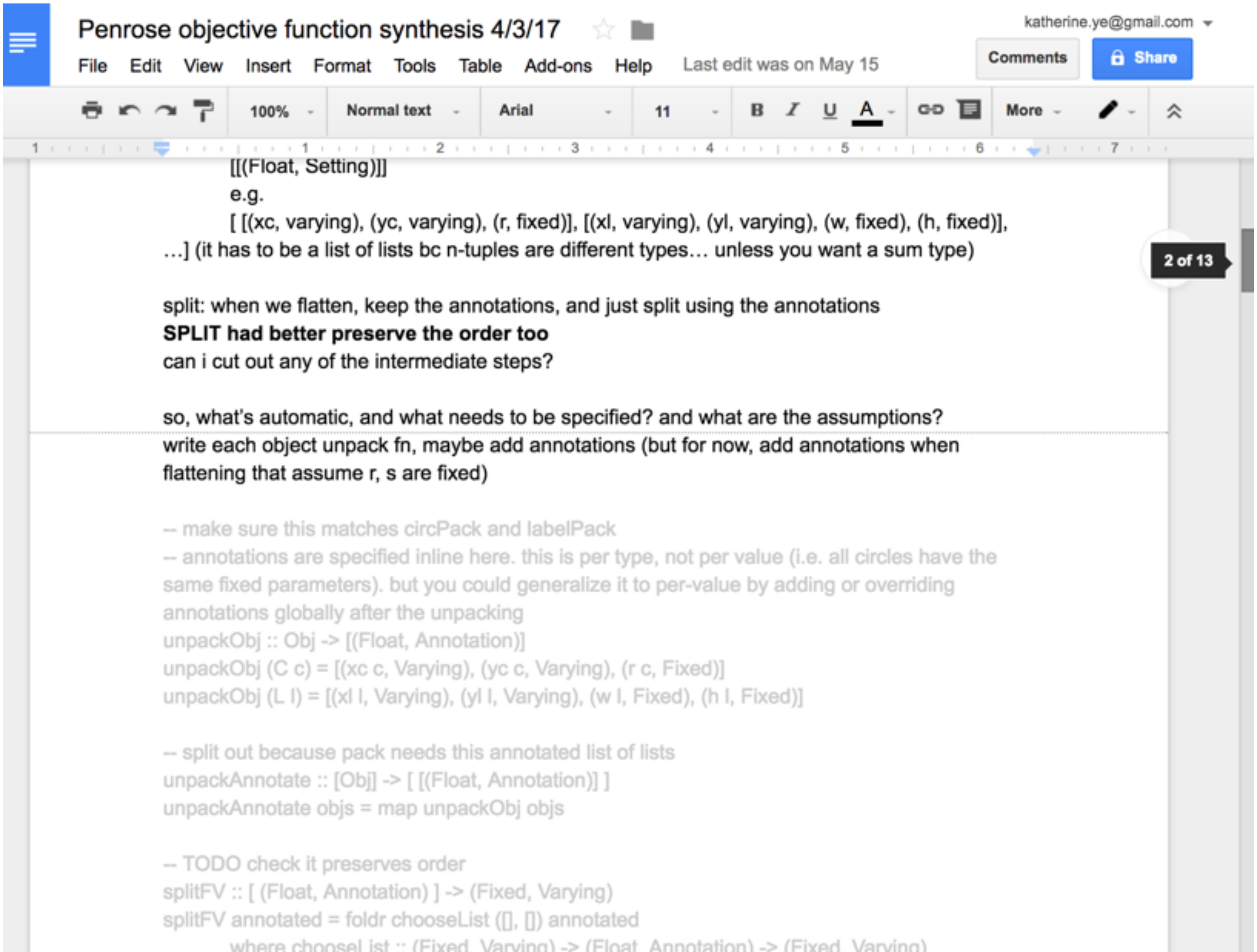
1.1 Contributions

DRBGs have the special property that testing, even sophisticated statistical fuzz-testing, cannot assure the security of an implementation. Even at the level of a pure cryptographic protocol, proving the security of a DRBG construction can be quite challenging. (Hybrid-game-based proofs are the only known technique, but for nontrivial DRBGs those proofs have so many technical steps that it is hard to trust them.) Therefore a new paradigm is required, in which the functional model of a DRBG is proved *with a machine-checked proof* to satisfy the appropriate PRF properties, and the C-language implementation (*and* its compilation to machine language) is proved

[1] A note on terminology: we use "entropy" loosely to denote randomness that is not predictable by an adversary. We use "sampled uniformly at random" and "ideally random" interchangeably. We use PRG, the acronym for "pseudo-random generator," to refer to the abstract cryptographic concept, whereas we use DRBG, the acronym for "deterministic random bit generator," to denote the specifications and implementations of PRGs. Instead of DRBG, some papers use "PRNG," the acronym for "pseudo-random number generator." The terms are synonymous.

Desire paths in programming interfaces:
that’s a whole new talk…

Spot the desire paths!



Penrose objective function synthesis 4/3/17

katherine.ye@gmail.com

File Edit View Insert Format Tools Table Add-ons Help Last edit was on May 15

Comments Share

100% Normal text Arial 11 B I U A More

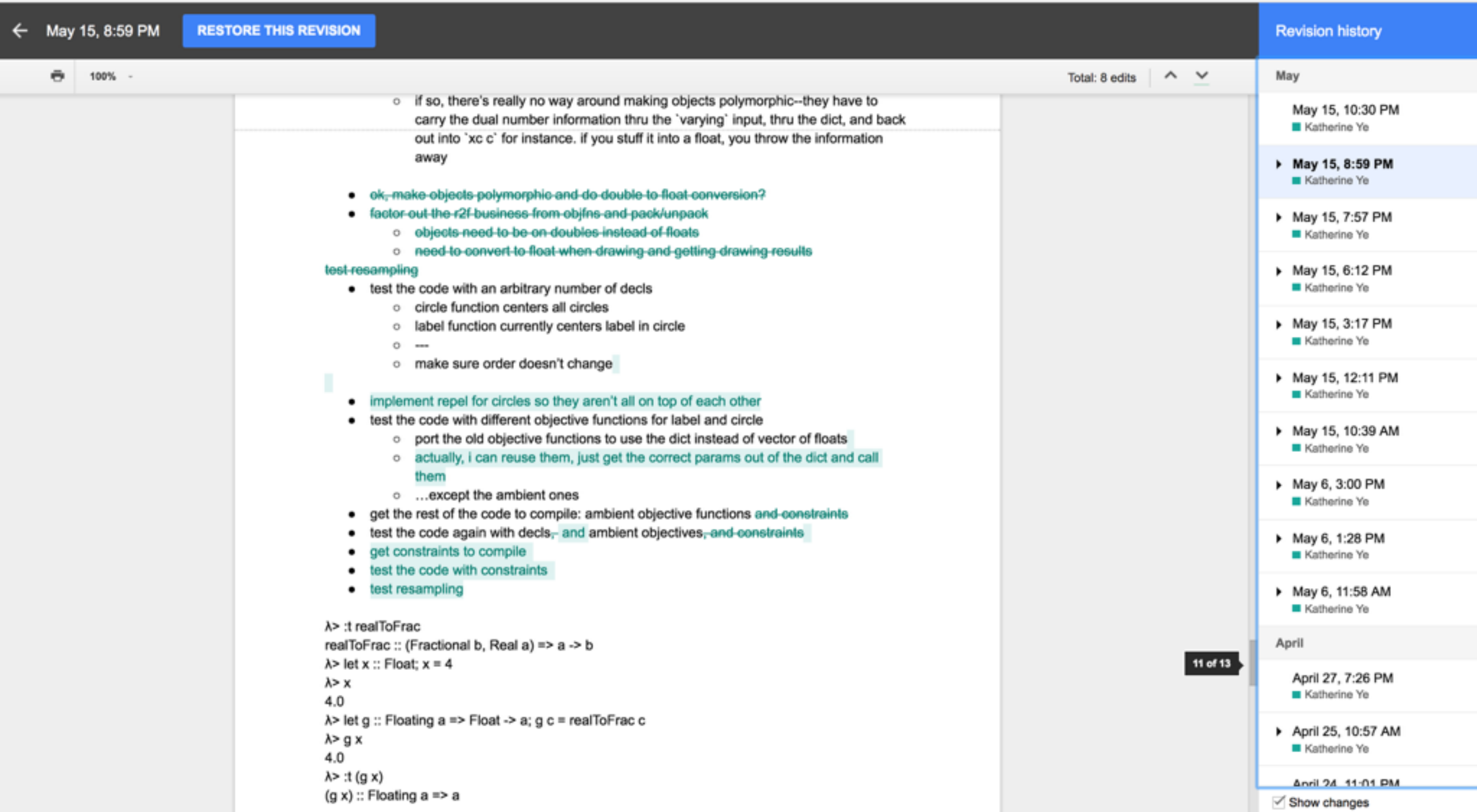
2 of 13

[[(Float, Setting)]]
e.g.
[[(xc, varying), (yc, varying), (r, fixed)], [(xl, varying), (yl, varying), (w, fixed), (h, fixed)], …] (it has to be a list of lists bc n-tuples are different types… unless you want a sum type)

split: when we flatten, keep the annotations, and just split using the annotations
SPLIT had better preserve the order too
can i cut out any of the intermediate steps?

so, what's automatic, and what needs to be specified? and what are the assumptions?
write each object unpack fn, maybe add annotations (but for now, add annotations when flattening that assume r, s are fixed)

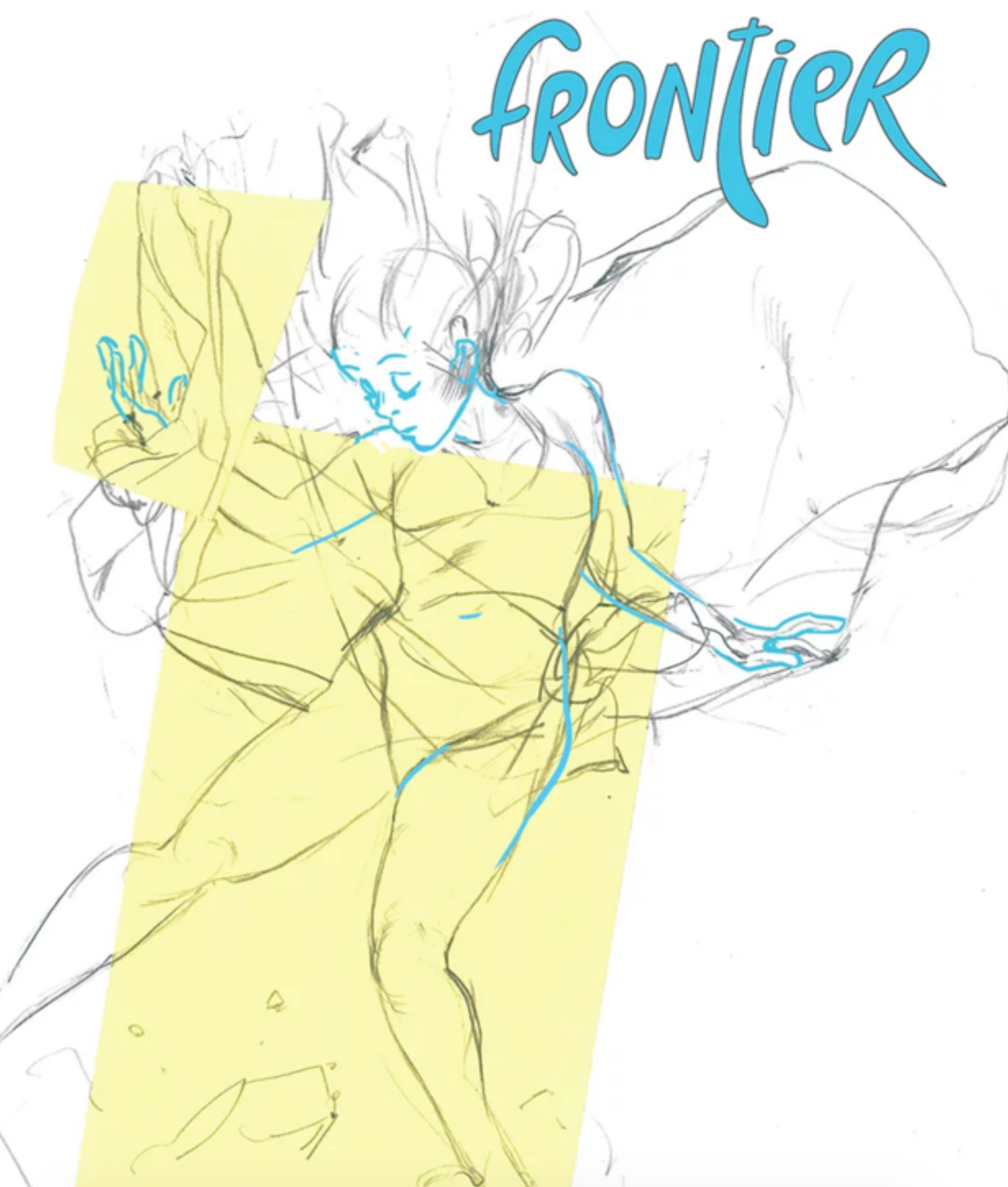
```
-- make sure this matches circPack and labelPack
-- annotations are specified inline here. this is per type, not per value (i.e. all circles have the same fixed parameters). but you could generalize it to per-value by adding or overriding annotations globally after the unpacking
unpackObj :: Obj -> [(Float, Annotation)]
unpackObj (C c) = [(xc c, Varying), (yc c, Varying), (r c, Fixed)]
unpackObj (L l) = [(xl l, Varying), (yl l, Varying), (w l, Fixed), (h l, Fixed)]

-- split out because pack needs this annotated list of lists
unpackAnnotate :: [Obj] -> [ [(Float, Annotation)] ]
unpackAnnotate objs = map unpackObj objs

-- TODO check it preserves order
splitFV :: [ (Float, Annotation) ] -> (Fixed, Varying)
splitFV annotated = foldr chooseList ([], []) annotated
        where chooseList :: (Fixed, Varying) -> (Float, Annotation) -> (Fixed, Varying)
```


May 15, 8:59 PM
RESTORE THIS REVISION
100%
Total: 8 edits
Revision history
if so, there's really no way around making objects polymorphic--they have to carry the dual number information thru the `varying` input, thru the dict, and back out into `xc c` for instance. if you stuff it into a float, you throw the information away
~~ok, make objects polymorphic and do double to float conversion?~~
~~factor out the r2f business from objfns and pack/unpack~~
~~objects need to be on doubles instead of floats~~
~~need to convert to float when drawing and getting drawing results~~
~~test resampling~~
test the code with an arbitrary number of decls
circle function centers all circles
label function currently centers label in circle
---
make sure order doesn't change
implement repel for circles so they aren't all on top of each other
test the code with different objective functions for label and circle
port the old objective functions to use the dict instead of vector of floats
actually, i can reuse them, just get the correct params out of the dict and call them
…except the ambient ones
get the rest of the code to compile: ambient objective functions ~~and constraints~~
test the code again with decls~~,~~ and ambient objectives~~, and constraints~~
get constraints to compile
test the code with constraints
test resampling
λ> :t realToFrac
realToFrac :: (Fractional b, Real a) => a -> b
λ> let x :: Float; x = 4
λ> x
4.0
λ> let g :: Floating a => Float -> a; g c = realToFrac c
λ> g x
4.0
λ> :t (g x)
(g x) :: Floating a => a
11 of 13
May
May 15, 10:30 PM
Katherine Ye
May 15, 8:59 PM
Katherine Ye
May 15, 7:57 PM
Katherine Ye
May 15, 6:12 PM
Katherine Ye
May 15, 3:17 PM
Katherine Ye
May 15, 12:11 PM
Katherine Ye
May 15, 10:39 AM
Katherine Ye
May 6, 3:00 PM
Katherine Ye
May 6, 1:28 PM
Katherine Ye
May 6, 11:58 AM
Katherine Ye
April
April 27, 7:26 PM
Katherine Ye
April 25, 10:57 AM
Katherine Ye
Show changes

How can we help humans turn prose into code?

Three findings

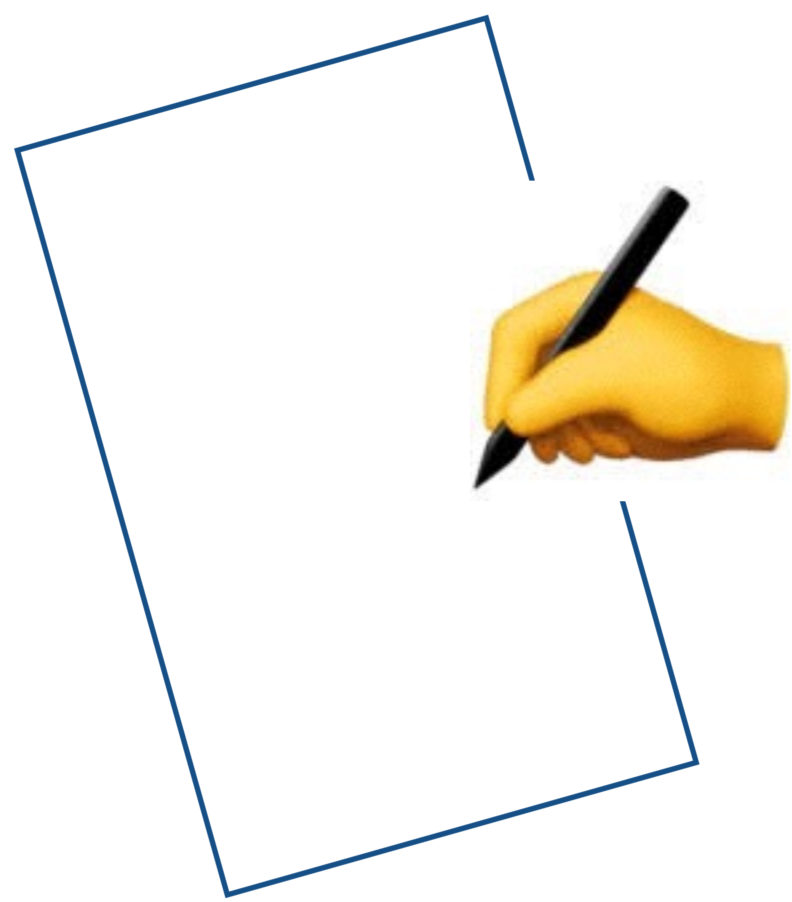
1. Many desire paths in creative interfaces are hacks to support process.



oh, here's the current build script

```
rm -rf build
mkdir -p build

# base concatenation
for each in *.md; do (cat $each; printf "\n\n---\n\n") >>
    build/base.md; done
# cat *.md > build/all.md

# convert all single returns into doubles so that pandoc doesn't complain
cat build/base.md | awk -v RS="" '{gsub (/\n\n/,"\n")}1' |
    awk -v RS="" '{gsub (/\n/,"\n\n")}1' >> build/all.md;

# create the .docx file
pandoc -f markdown_github -t docx -o build/all.docx build/all.md
```

2. The natural evolution of creative interfaces over time has been to support process.



3. We have a unique opportunity to reënvision interfaces to treat creative processes as first-class citizens.

The act of writing inevitably changes what you want to say…

[It’s] a process, not just of setting down a concept already fully formed in the mind, but of **discovering what it is you meant in the first place.**

First Draft of the Revolution
Emily Short & Liza Daly

a. Plunder from every timeline.

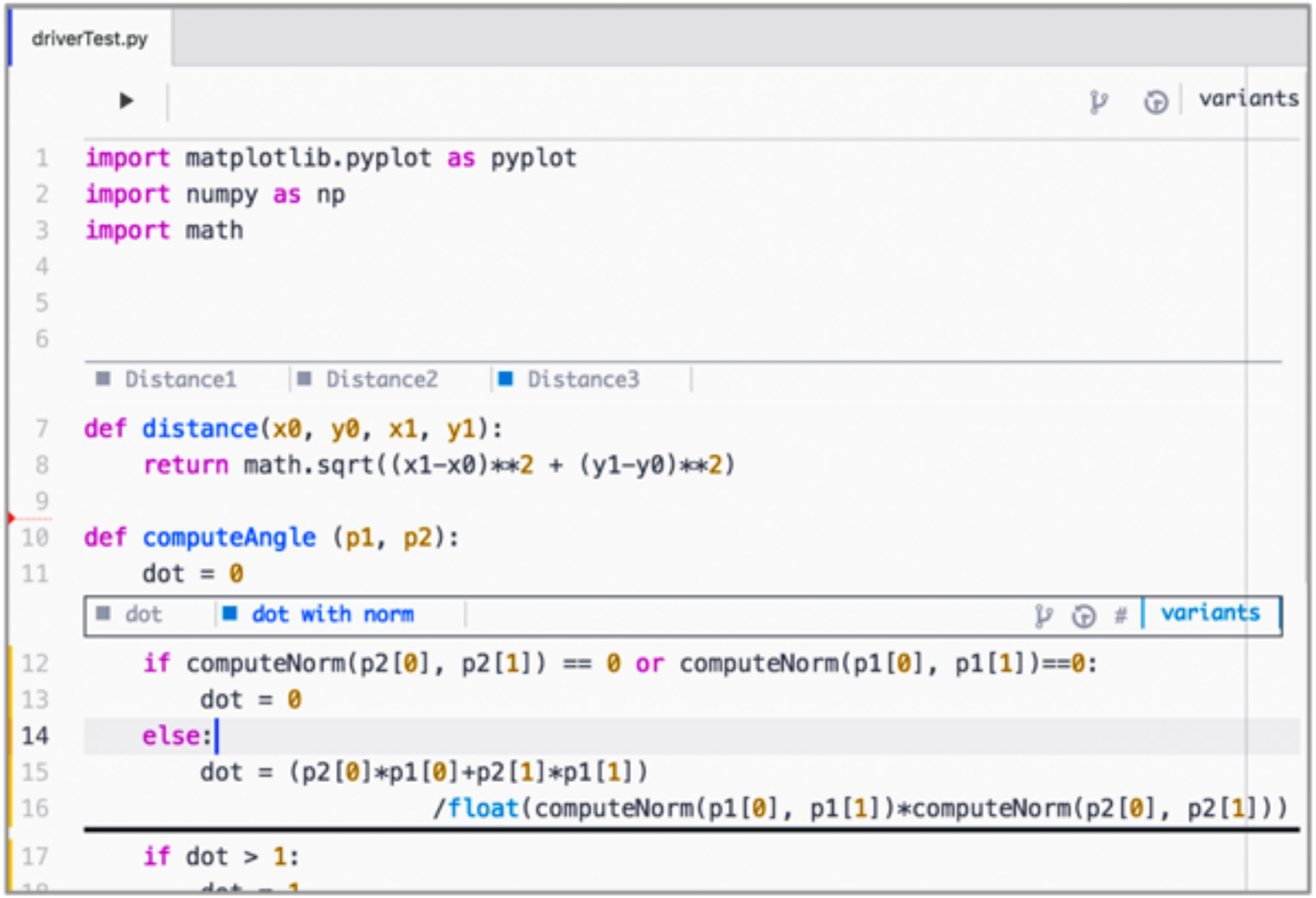
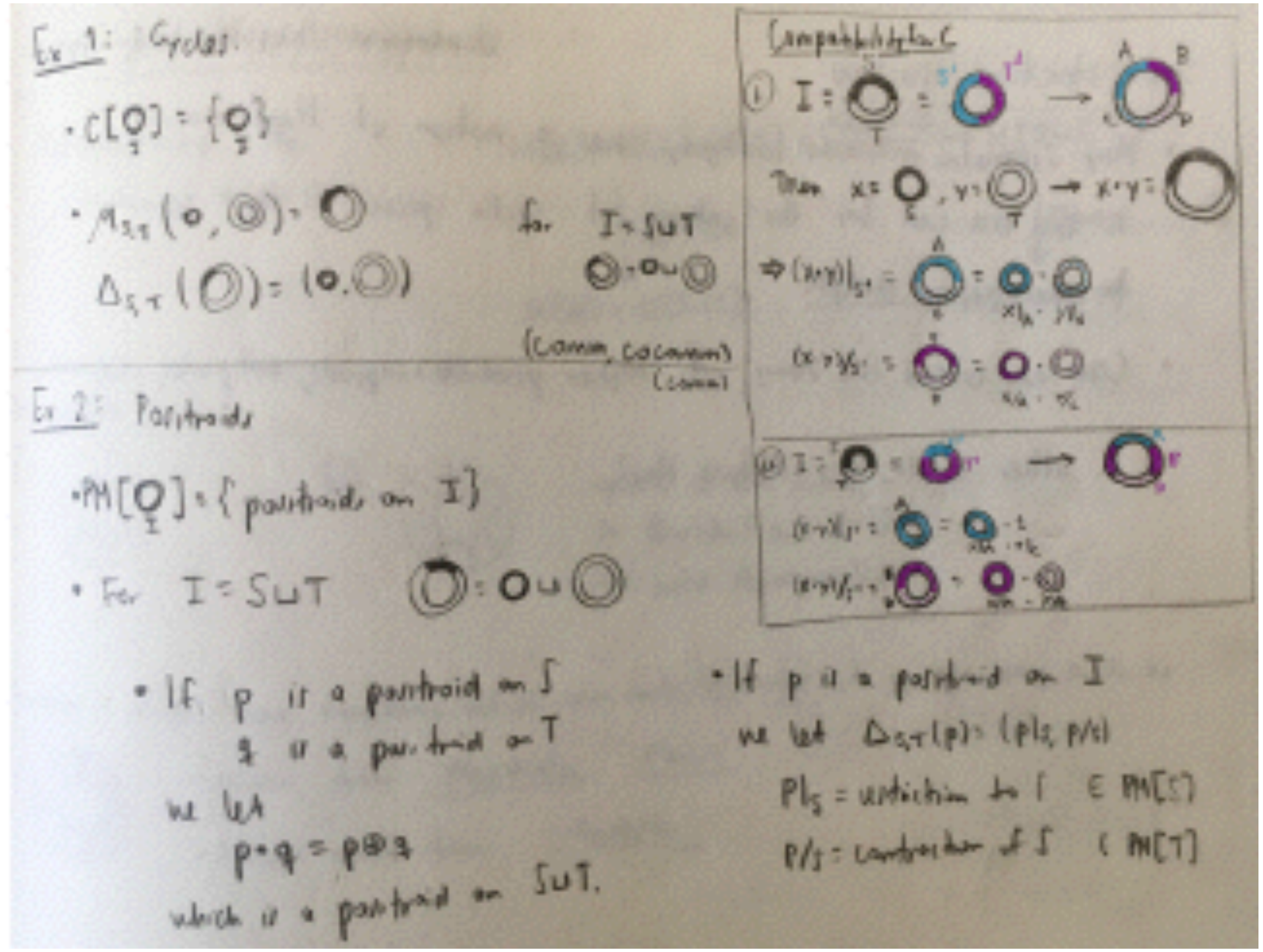


Figure 1. Variolite is a code editing tool that includes local versioning of chunks of code. Here, there are two version boxes. The outer one has three "Distance" versions, and the inner one has two "dot" versions with "dot with norm" currently being used.

b. Support the feedback loop between messy thought and neat thought.



from the notebook of Federico Ardila
h/t *Messy thought, neat thought*
May-Li Khoe

I'm building a tool to support mathematical process! [*http://penrose.ink*](http://penrose.ink)

How lovely would it be if our UI could help capture messiness and iteration in a way that made it visible and actively encouraged?

Messy thought, neat thought
May-Li Khoe

Running Incomplete Programs

Ian Voysey[1] Cyrus Omar[1] Matthew A. Hammer[2]

[1]Carnegie Mellon University (USA)
{iev, comar}@cs.cmu.edu

[2]University of Colorado Boulder (USA)
matthew.hammer@colorado.edu

applied to every list element uniformly. She might arrive at the incomplete term

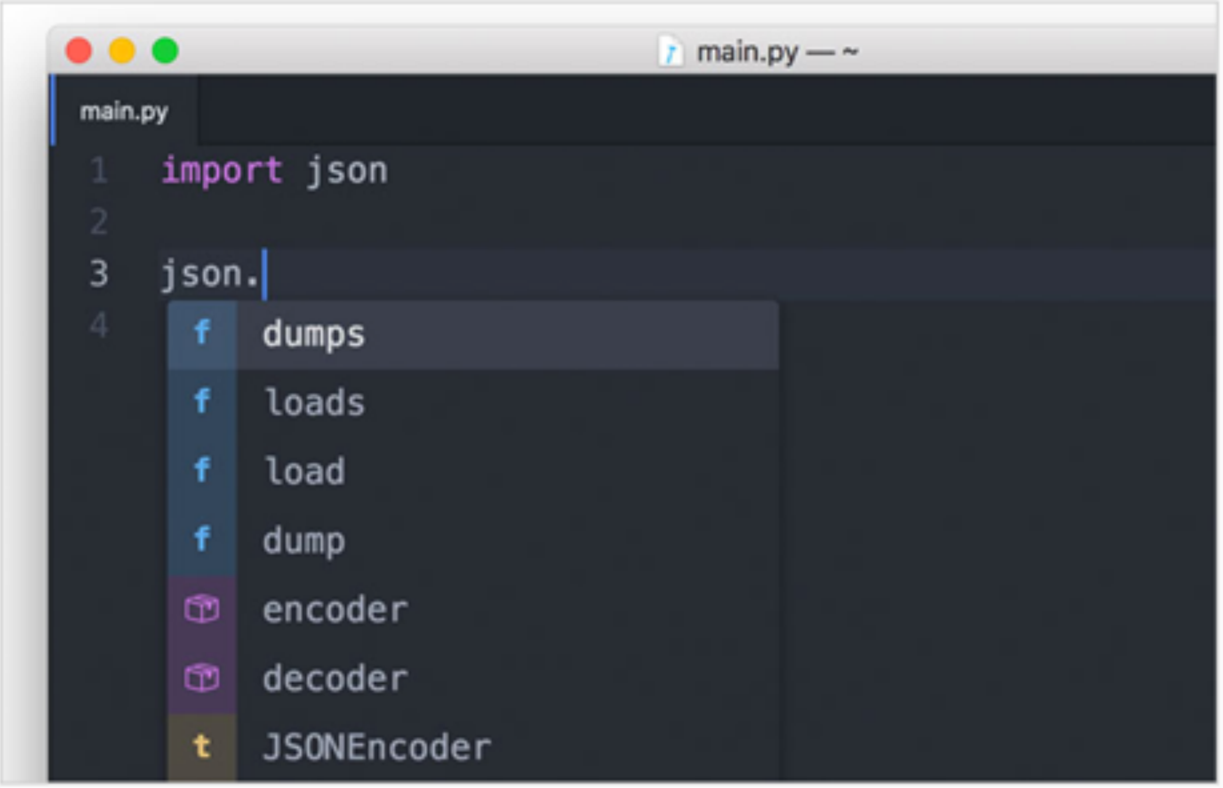
```
fun map f [] = []
  | map f (x::xs) = (|)u::(map f l)
```

c. Build in access to humanity's prior work: search, synthesis, remixing.

mcc
@mcclure111

I feel like I learned something about UX reading this

JP LeBreton @vectorpoem
Doom 2 map01 was remixed so heavily partly because in many editors it was the first thing you saw on startup. Way better than a blank page.



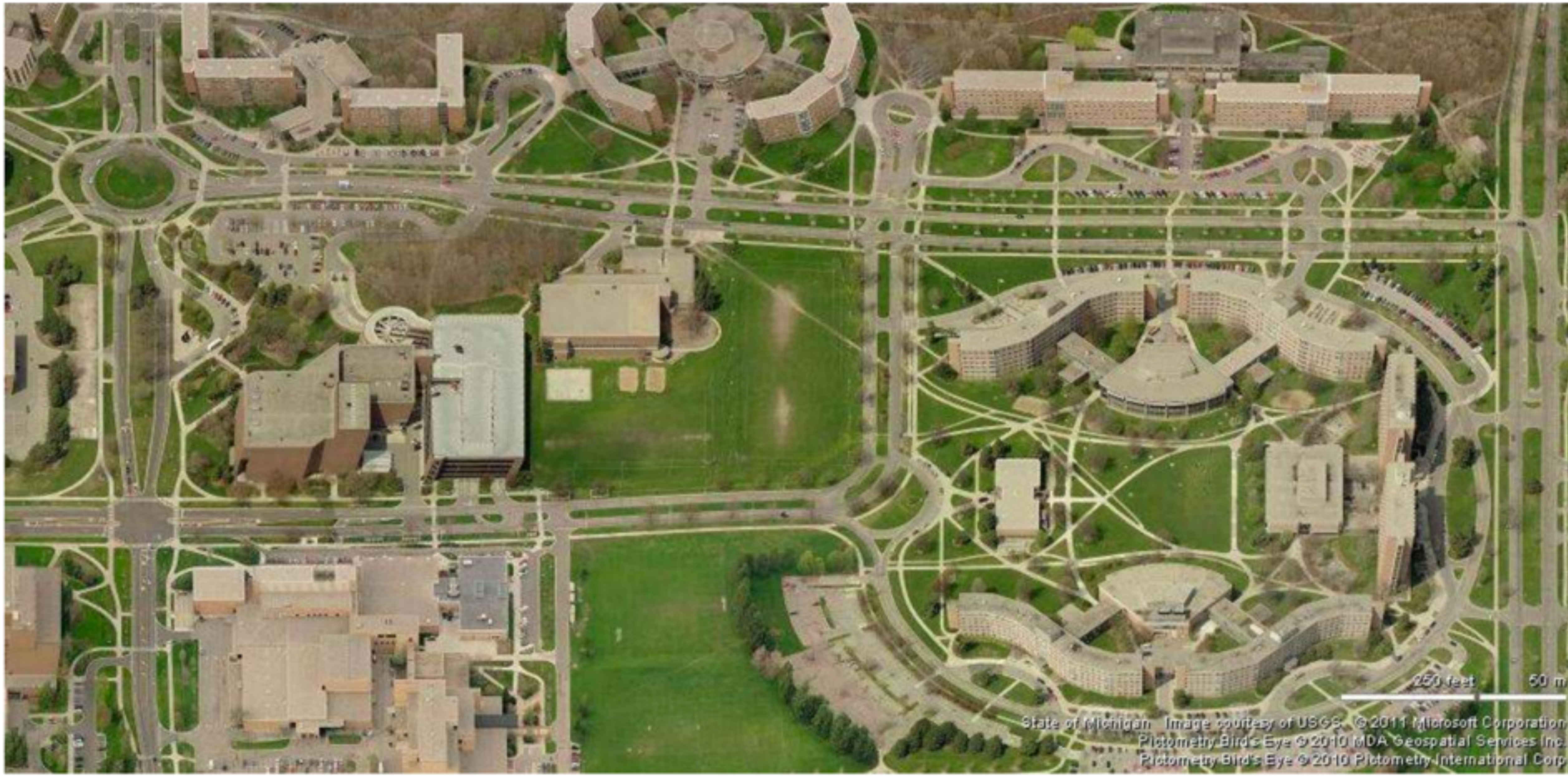
Ranked completions keep you in flow

Kite's code completions are powered by the best Python type inference engine available. Kite analyzes all the code on the web and gives you fast, smart completions ordered by popularity data, not the alphabet.

AI-assisted design: also a whole new talk.

See Hebron's "Rethinking design tools in the age of machine learning."

Let’s reify high-level processes!



Emergent desire paths turned into sidewalks over time at Michigan State University

http://99percentinvisible.org/article/least-resistance-desire-paths-can-lead-better-design/

FEATURES

Markdown & Preview

Using simple punctuation to format your text, lets you keep your hands on the keyboard so you can just write.

Live Sync

supports a multitude of storage services and platforms for syncing your documents between devices in real-time.

Document Library

In , swipe from the left to show the document Library. Search, sort, and quickly swap between documents while focusing on the same window.

Syntax Highlight

Syntax Control improves upon Focus Mode by helping you to gain insight into your text's grammatical structure.

File Export

for Mac lets you export a document to HTML, Microsoft Word (.docx), and PDF formats.

Custom Template

This is what you have been waiting for. You can now create custom templates for preview, printing and PDF export.

Focus Mode

Focus Mode dims everything but the current sentence, helping you stay in the flow, get words out, and avoid the temptation to edit.

Night Mode

includes an inverted "light on dark" mode, perfect for writing at night.

Content Blocks

With 4 it is now possible to embed pictures, tables and text through file transclusion.

FEATURES

Rich history model

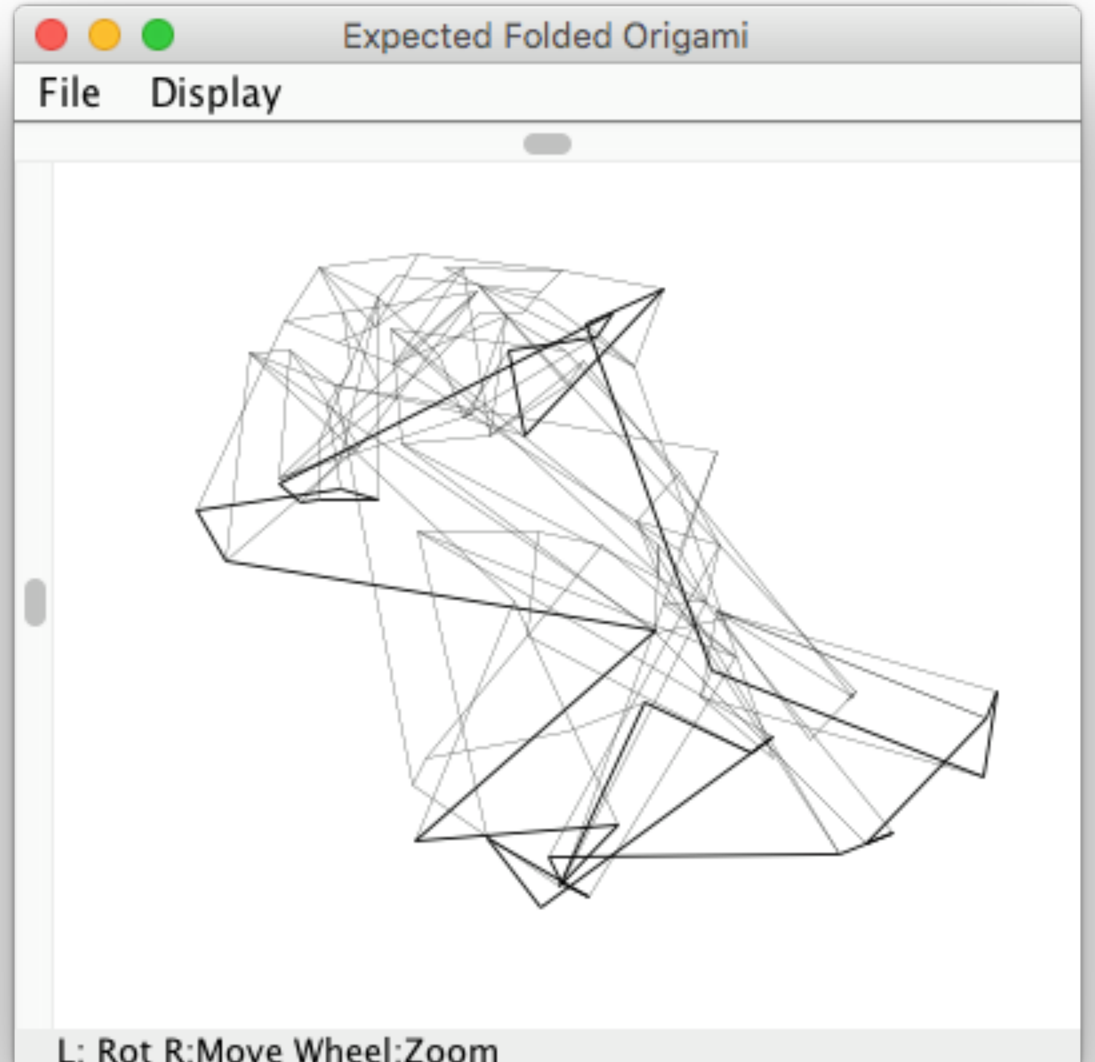
The undo tree model enables you to retain all history: work without fear, and seamlessly switch between and edit branches.

Live process visualization

Visualize your work's change over time and make a heatmap of most edited regions, all in real time.

Intelligent, controllable diffing

We can infer changesets, or you can set semantic patches for experimentation, and easily swap between them and take notes.



Thanks!

svn.ink/process

@hypotext

*Want to collaborate on process?
Let's talk!*

Thanks to Angela Zhou, Raymond Zhong, Chris Beiser, Robert Ochshorn, Nate Sauder, Sherjil Ozair, and Vrushank Vora for helpful feedback.

Discussion

- What are your own processes and process artifacts?
- What are your desire paths in interfaces, process-related or not?
- What are the most *anti-process* interfaces?
- Jam suggested by Max Kreminski: talk to your neighbor and create a tiny tool to support some part of their creative process.
- Zines: I will mail you a physical copy. Email me your address! kqy@cs.cmu.edu

Appendix

Comments from audience

- (Apologies if I misremembered your response!)
- "We don't have to just support desire paths. We can, and should, show people whole new ways of doing things."
- "Work by Klemmer et al. shows that a group that is forced to maintain separate iterations tends to create better work than a group that must iterate in one thread." (?) (I may have misquoted)
- "This reminds me of the time that Sketch removed the ability to export nested artboards. Designers store iterations in these artboards, and they got so angry on Twitter that Sketch put the feature back."
- "I personally doubt that any software could support all the processes I want. I need to move between mediums: physically move from a 'creation' space to a 'refinement' space."
- "I prefer working on typewriters and speaking on phones because otherwise I can spend endless time fiddling with the process."
- "I wonder about the relation of this work to that of encouraging disfluencies such as 'um,' 'er,' and 'ah' in conversation."

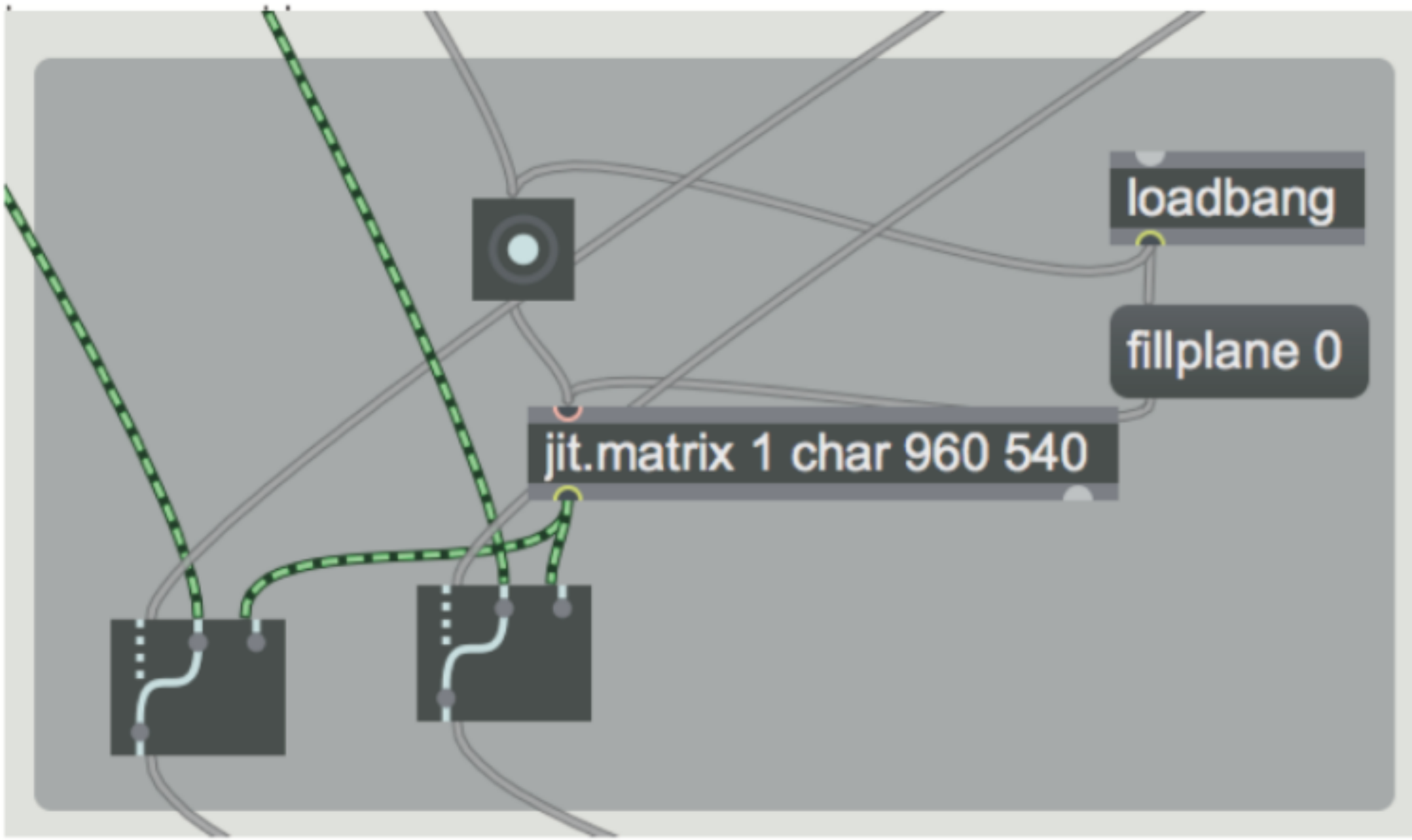
Ideas from Nate Sauder:

Write an emacs hook where each save automatically creates a commit, then squash manually. Writing the commit message is very helpful. You can recursively squash. git should support hierarchical commits!

See also magit and git-messenger: can have one file displaying 4-5 versions at a time.

Piece by Avneesh Sarwate

This is an inactive section for a feature that didn't end up working by the time the performance happened. Such is art under deadlines.



François Chollet

@fchollet

Humans think in terms of sequences of actions. Arguably the best way to emulate human creative behaviors is by generating data sequentially.

Footnote:

The reason I dislike the question “can computers make art?” is that it destroys the human creative process. The process is so powerful: modeled sequentially, it helps you discover what you wanted in the first place. You don’t just generate an image. I think a better way is for computers to collaborate in helping a human discover what it is they wanted in the first place.

obligatory

tweak slides, adding notes from topos practice talk
katherineye committed 34 seconds ago

Commits on Jun 9, 2017

finish rough talk w/ ending
katherineye committed 2 hours ago

add section for 'first draft of the rev' case study; edit case study ... ...
katherineye committed 15 hours ago

significantly reorganize talk to move hypotheses/reificataion to the ... ...
katherineye committed 17 hours ago

Commits on Jun 8, 2017

finish zine intro. how to segue into 'creative interfaces'?
katherineye committed a day ago

add writing diff/intensity combined
katherineye committed 2 days ago

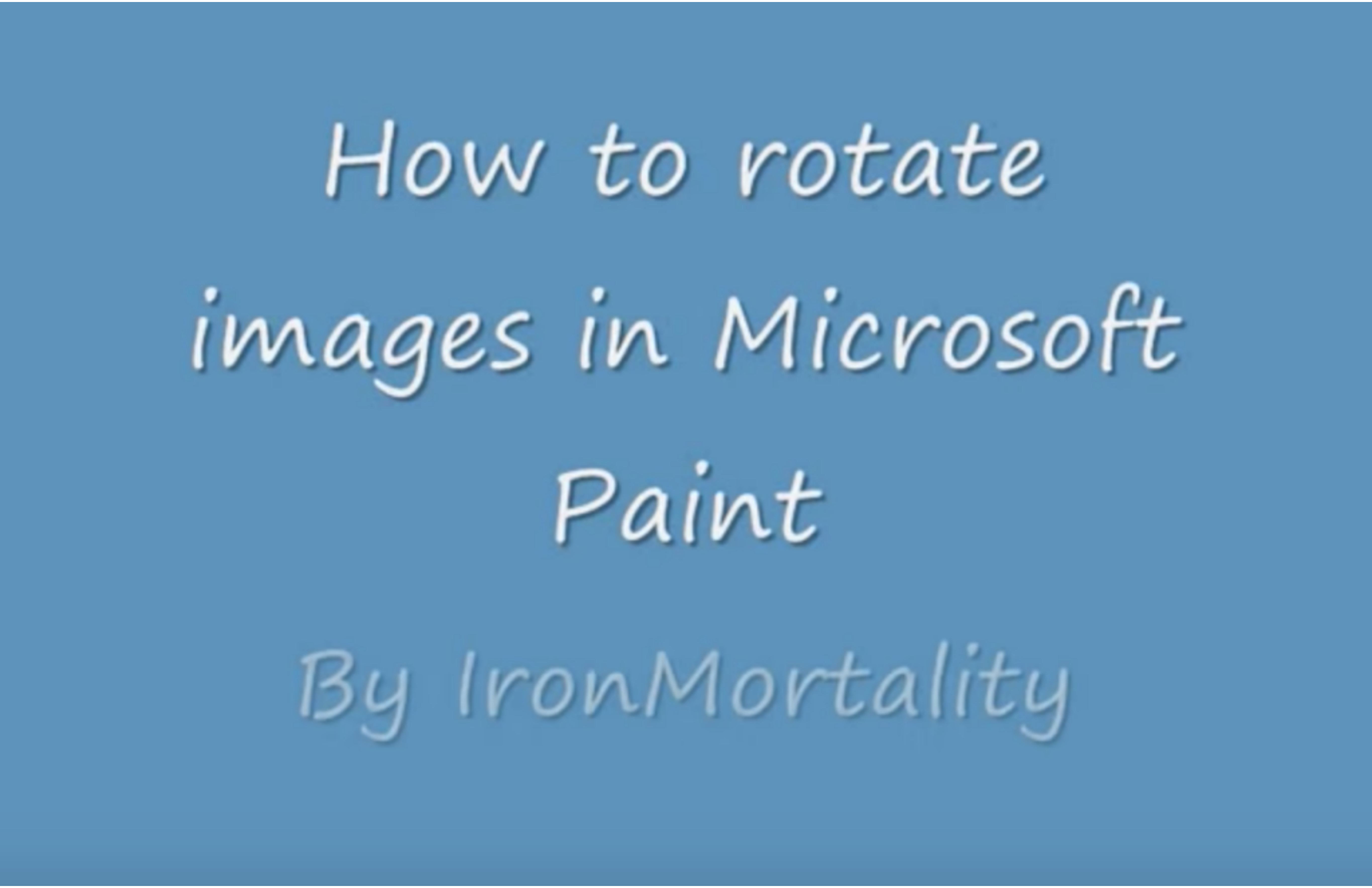
add most important part: backtracking and alternative layering in dra...
...
katherineye committed 2 days ago

add flow for postits (p. happy with it) and add booklet (couldn't print)
katherineye committed 2 days ago

Commits on Jun 4, 2017

add a few more slides; in gdocs, work thru concrete interface example...
...
katherineye committed 5 days ago

start talk: throw in rough headings and screenshots
katherineye committed 6 days ago



https://www.youtube.com/watch?v=wUKMIi0mInc via Chris Beiser

References

- PROCESS I: Angela, Katherine, Raymond
- PROCESS II: Lea, Chris, Neeta, Diana, Max, and Avneesh's pieces and comments
- "Least resistance: how desire paths can lead to better design" (Kurt Hohlstedt and photographers at 99% Invisible)
- *Thought as a technology* (Michael Nielsen)
- Frontier #14: Rebecca Sugar (Youth in Decline)
- *First Draft of the Revolution* (Emily Short and Liza Daly)
- "Variolite: supporting exploratory programing by data scientists" (Kery and Myers)
- "Running incomplete programs" (Voysey et al.)
- "Messy thought, neat thought" (May-Li Khoe, Khan Academy Long-Term Research)
- "Causality: a conceptual model of interaction history" (Nancel and Cockburn)
- "Integrating Prose as First-Class Citizens with Models and Code" (Voelter)
- "Mosaic: designing online creative communities for sharing works-in-progress" (Kim et al.)
- "Generative visual manipulation on the visual manifold" (Zhu et al.)
- "Rethinking design tools in the age of machine learning" (Patrick Hebron)
- Related: distributed and embodied cognition
- Max Kreminski's tweet on making microtools to support someone else's creative process
- François Chollet's tweet on creativity as a time-based process
- @mcclure and @vectorpoem's tweets on a creative initial state
- Software cited: Photoshop, Illustrator, iA Writer, git, emacs, Scrivener, Agda, Kite