Gadget

Design Values

2

This document sets out the guiding design values for the Gadget project. These are drawn from the survey of related works, and articulate the design philosophies and ideas we think are the most relevant to Gadget.

Five high level design values were distilled from the review: a quality of world-ness, linked representation, tactile, personally meaningful, and directed and undirected activity.

In the next stage of the project—stage 3—we will design and prototype various ideas, and draw directly upon the guiding ideas outlined here.

Chaim Gingold • Wed Jul 05 2017

Gadget

Purpose. The Gadget project combines cutting edge ideas from programming languages and game design to invent new tools for novices learning to code as well as expert users. It is predicated on the observation that some of the most powerful ideas in the history of computers—from interface design to programming languages—have come from making systems more tangible, alive, playful, and accessible to children. Drawing on influences from Smalltalk to Minecraft, Gadget seeks to build captivating play experiences that transform users into proficient and creative computational thinkers. But Gadget is more than a playful tutorial; it aims to transform the experience of programming itself.

Approach. The project is divided into four stages: survey, articulating design
1 values, prototyping, and design. In the first stage, we conducted an open-minded
and thorough survey of related work, from Smalltalk to Minecraft. We surveyed 48
2 different systems, producing one page visual distillations for each. In stage two, five
high level design values were distilled from the review: a quality of world-ness,
linked representation, tactile, personally meaningful, and directed and undirected
3 activity. In the third stage of the project, we will put these design values into action
by building and testing prototypes that push the envelope in programming
4 environment design. In the fourth stage we will summarize our learnings in the form
of a design for a new computational world.

Who. The Gadget project builds upon the combined background and expertise of Dan Ingalls and Chaim Gingold. Among his many seminal contributions to computing, Dan Ingalls has contributed to making programming more tangible, alive, and open to creative improvisation (e.g. Squeak and Lively). Chaim Gingold brings to the project expertise in designing simulations, play experiences, and creative tools. Recent projects include using simulation toys as book illustrations (Earth Primer), and investigations into diagrammatic representations of software (Ph.D. dissertation).

Gadget

Design Values

World

Linked Representations

Tactile

Meaningful

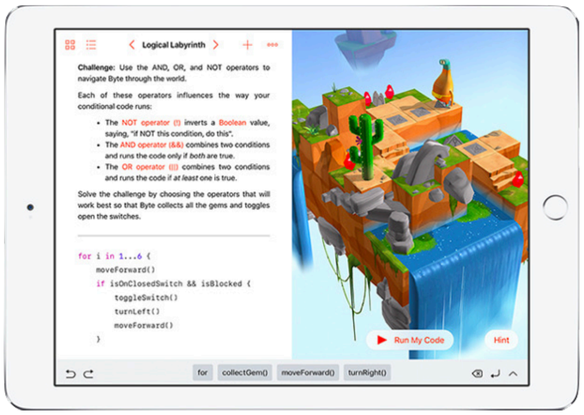
Directed & Undirected

World

Compelling and computational worlds draw people in.

Appealing

Appealing worlds draw us in with aliveness, charm, and immersion.



Challenge: Use the AND, OR, and NOT operators to navigate Byte through the world.

Each of these operators influences the way your conditional code runs:

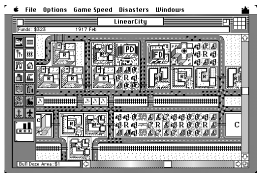
- The NOT operator (!) inverts a Boolean value, saying, "if NOT this condition, do this".
- The AND operator (&&) combines two conditions and runs the code only if *both* are true.
- The OR operator (||) combines two conditions and runs the code if *at least* one is true.

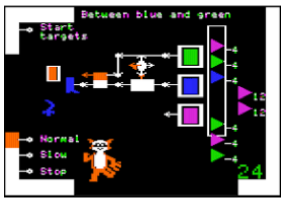
Solve the challenge by choosing the operators that will work best so that Byte collects all the gems and toggles open the switches.

```
for i in 1...6 {
    moveForward()
    if isOnClosedSwitch && isBlocked {
        toggleSwitch()
        turnLeft()
        moveForward()
    }
```

Computational

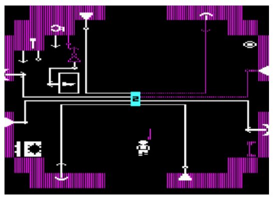
Some worlds are made of stateful computational elements that interact with one another. Cellular automata and CA like systems such as *SimCity* and *Minecraft* are prime examples of computational worlds that clearly show their algorithmic machinations.

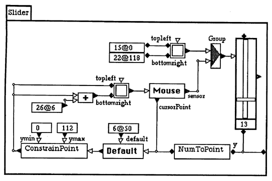


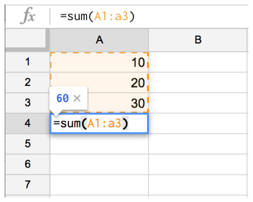


Spatial

Some worlds offer concrete spatial relationships between elements, and spaces to traverse. More concretely fleshed out worlds exhibit a quality of "naive realism" (diSessa 1985).

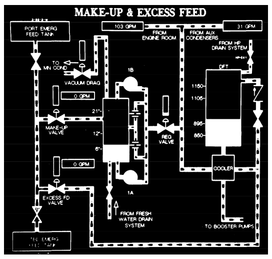






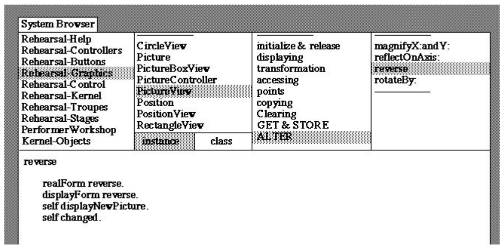
=sum(A1:a3)

	A	B
1	10	
2	20	
3	60 × 30	
4	=sum(A1:a3)	
5		
6		
7		



Explore, Build, and Inhabit

Worlds can be places to explore, tinker with, build, and take up residence in.



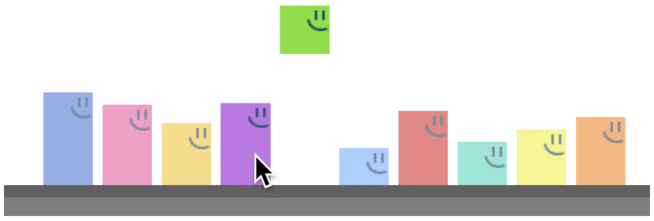
System Browser

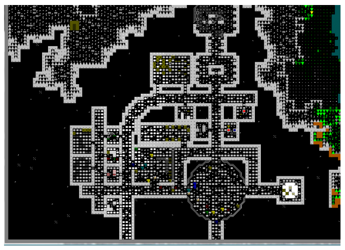
Rehearsal-Help	CircleView	initialize & release	magnifyX:andY:
Rehearsal-Controllers	Picture	displaying	reflectOnAxis:
Rehearsal-Buttons	PictureBoxView	transformation	reverse
Rehearsal-Graphics	PictureController	accessing	rotateBy:
Rehearsal-Control	PictureView	points	
Rehearsal-Kernel	Position	copying	
Rehearsal-Troupes	PositionView	Clearing	
Rehearsal-Stages	RectangleView	GET & STORE	
PerformerWorkshop	instance / class	ALTER	
Kernel-Objects			

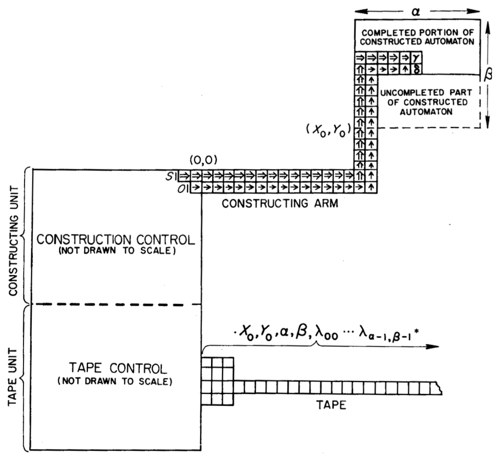
```
reverse

    realForm reverse.
    displayForm reverse.
    self displayNewPicture.
    self changed.
```

World as Data

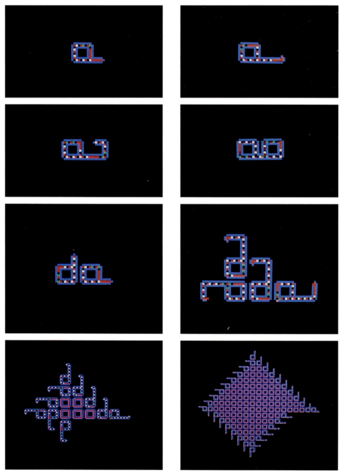
When every part of a computational world is concretely manifested it takes on the quality of *world as data*. Everything that is part of the world For example, Glen Chiacchieri's *Bubble Sort* (2016), *Minecraft, Dwarf Fortress*, and cellular automata inspired systems. This also increases the tactility of a system.





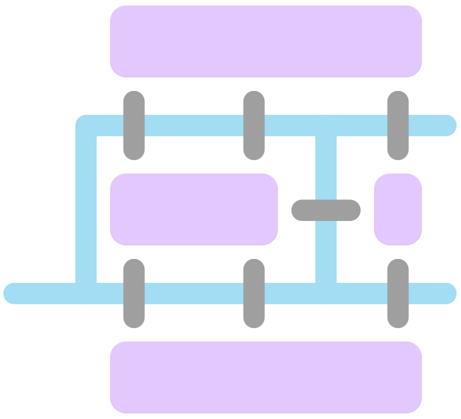


α
COMPLETED PORTION OF CONSTRUCTED AUTOMATON
β
UNCOMPLETED PART OF CONSTRUCTED AUTOMATON
(X_0, Y_0)
(0,0)
CONSTRUCTING ARM
CONSTRUCTING UNIT
CONSTRUCTION CONTROL (NOT DRAWN TO SCALE)
TAPE UNIT
TAPE CONTROL (NOT DRAWN TO SCALE)
$X_0, Y_0, \alpha, \beta, \lambda_{00} \cdots \lambda_{\alpha-1,\beta-1}{}^{*}$
TAPE



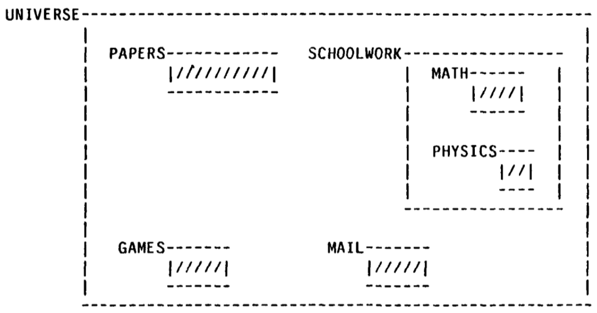
Euler's Seven Bridges of Königsberg problem set the stage for graph theory, functioning as an appealing, spatial, and explorable world for an abstract problem:

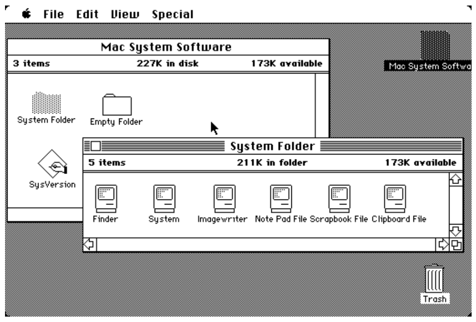
KONINGSBERGA

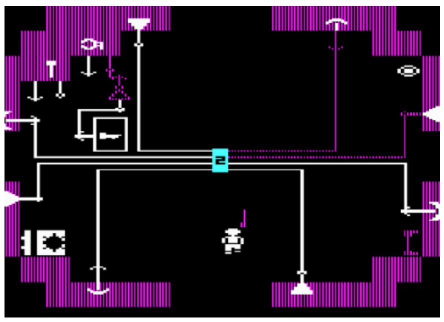


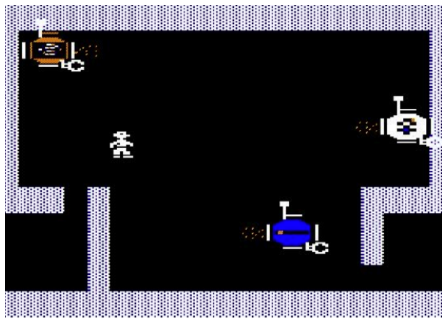
Zoom and Encapsulation

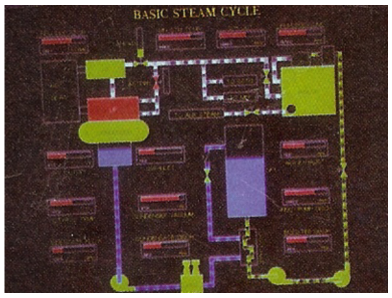
By nesting spaces within one another, complex world can be built up that encapsulate and interlink multiple representations.

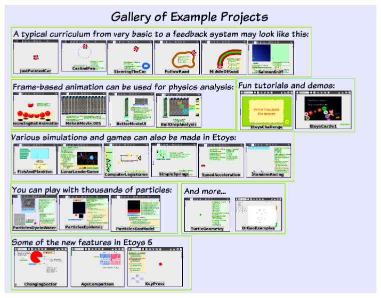


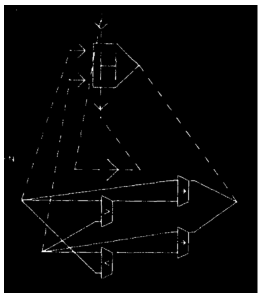


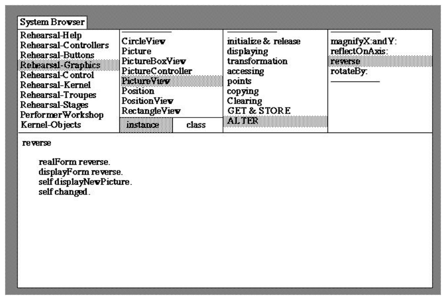




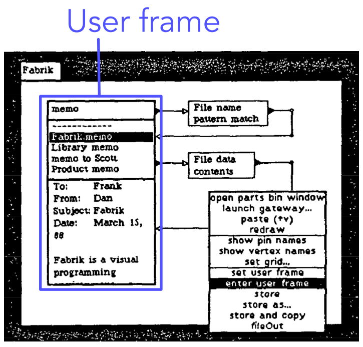



Gallery of Example Projects
A typical curriculum from very basic to a feedback system may look like this:
Frame-based animation can be used for physics analysis:
Fun tutorials and demos:
Various simulations and games can also be made in Etoys:
You can play with thousands of particles:
And more...
Some of the new features in Etoys 5





```
reverse

    realForm reverse.
    displayForm reverse.
    self displayNewPicture.
    self changed.
```



To: Frank
From: Dan
Subject: Fabrik
Date: March 15, 88

Fabrik is a visual programming

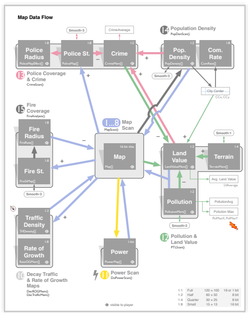
Shown above: On-Line Graphical Specification of Computer Procedures (Sutherland 1966), Smalltalk (1976), Boxer (diSessa 1982), Macintosh (1984), Steamer (Hollan et al. 1984), Robot Odyssey (Wallace and Grimm 1984), and Fabrik (Ingalls 1988), Squeak/EToys (1996).

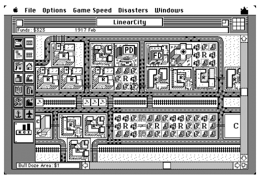
Linked Representations

Connecting multiple perspectives helps people find powerful new ways to see, think, and operate.

Multiple Abstractions

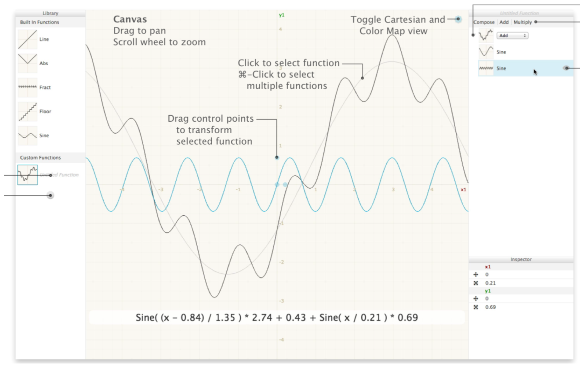
Multiple representations, such as overviews, zooming, and layers of abstractions, afford multiple perspectives, ways of thinking, and pathways to mastery.



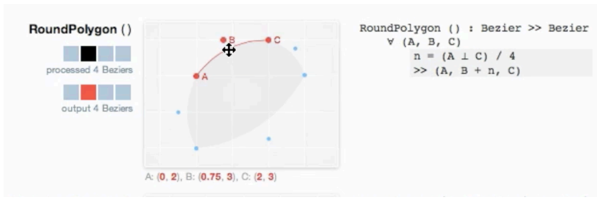


Coupled Representations

Coupling multiple representations helps us to fluidly move between them. Begin with the representation that makes sense to you, and use it to gain traction in alternate ways of seeing and thinking.



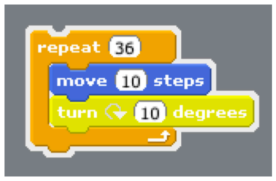
Sine((x – 0.84) / 1.35) * 2.74 + 0.43 + Sine(x / 0.21) * 0.69

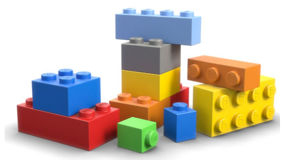


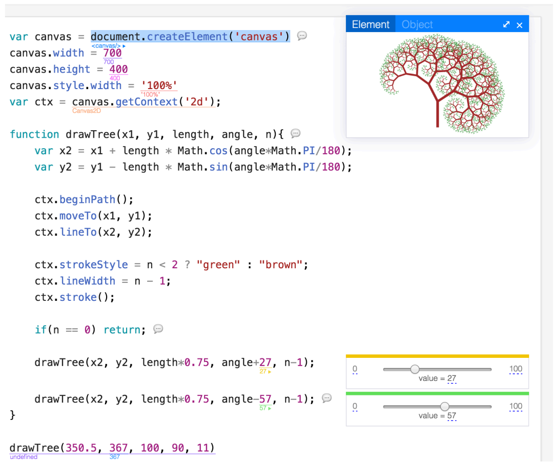
```
RoundPolygon () : Bezier >> Bezier
    ∀ (A, B, C)
        n = (A ⊥ C) / 4
        >> (A, B + n, C)
```

Schematic

As in a toy or diagram, simplicity and transparency make representations clear and easy to absorb.



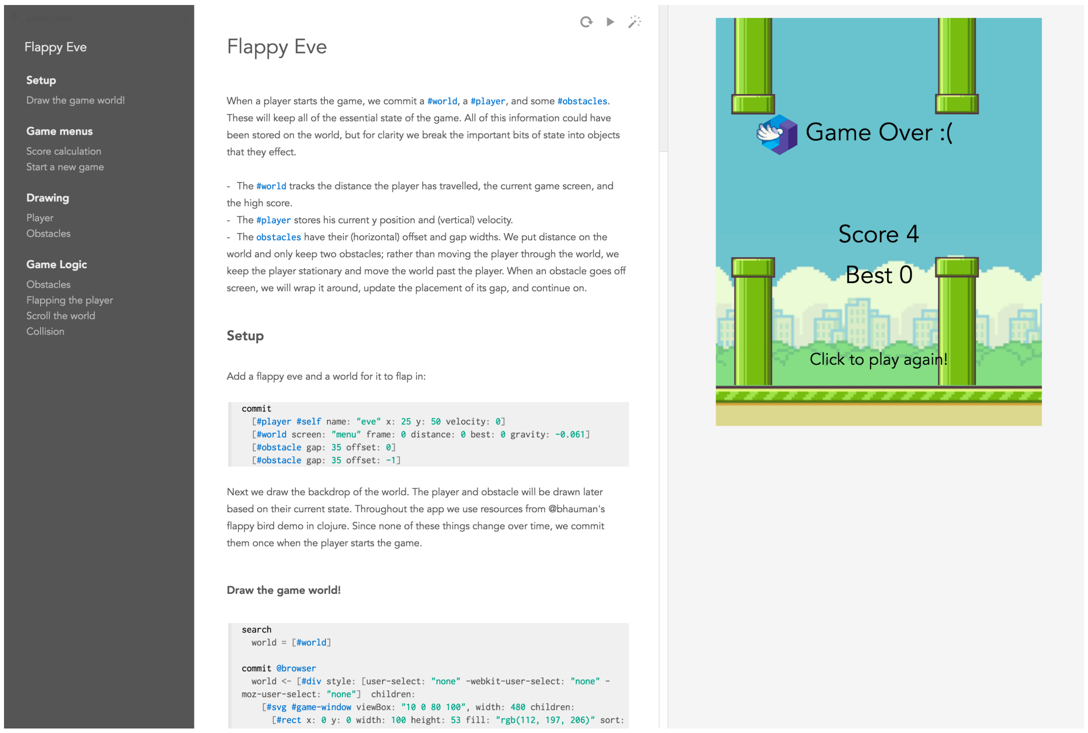




```
var canvas = document.createElement('canvas')
canvas.width = 700
canvas.height = 400
canvas.style.width = '100%'
var ctx = canvas.getContext('2d');

function drawTree(x1, y1, length, angle, n){
    var x2 = x1 + length * Math.cos(angle*Math.PI/180);
    var y2 = y1 - length * Math.sin(angle*Math.PI/180);

    ctx.beginPath();
    ctx.moveTo(x1, y1);
    ctx.lineTo(x2, y2);

    ctx.strokeStyle = n < 2 ? "green" : "brown";
    ctx.lineWidth = n - 1;
    ctx.stroke();

    if(n == 0) return;

    drawTree(x2, y2, length*0.75, angle+27, n-1);

    drawTree(x2, y2, length*0.75, angle-57, n-1);
}

drawTree(350.5, 367, 100, 90, 11)
```

Hyperliteracy

In literate programming the flow of a programmer's thoughts is the primary textual narrative; the details of program logic is subordinate. Literate programs traditionally look like prose interspersed with program code. Eve, shown below, exhibits what I describe as *hyperliterate programming*. Not only are prose and program interspersed (center column of illustration), but so is the higher level structure of program architecture (left column). Multiple representations, at multiple levels of abstraction, of a program are interlinked.



Flappy Eve

When a player starts the game, we commit a #world, a #player, and some #obstacles. These will keep all of the essential state of the game. All of this information could have been stored on the world, but for clarity we break the important bits of state into objects that they effect.

- The #world tracks the distance the player has travelled, the current game screen, and the high score.
- The #player stores his current y position and (vertical) velocity.
- The obstacles have their (horizontal) offset and gap widths. We put distance on the world and only keep two obstacles; rather than moving the player through the world, we keep the player stationary and move the world past the player. When an obstacle goes off screen, we will wrap it around, update the placement of its gap, and continue on.

Setup

Add a flappy eve and a world for it to flap in:

```
commit
  [#player #self name: "eve" x: 25 y: 50 velocity: 0]
  [#world screen: "menu" frame: 0 distance: 0 best: 0 gravity: -0.061]
  [#obstacle gap: 35 offset: 0]
  [#obstacle gap: 35 offset: -1]
```

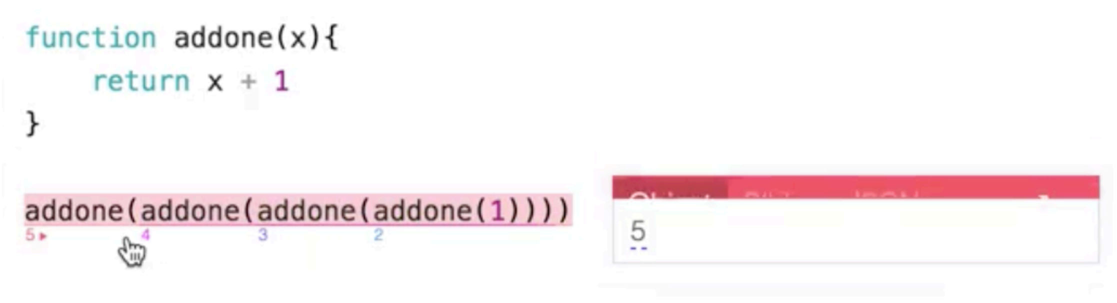
Next we draw the backdrop of the world. The player and obstacle will be drawn later based on their current state. Throughout the app we use resources from @bhauman's flappy bird demo in clojure. Since none of these things change over time, we commit them once when the player starts the game.

Draw the game world!

```
search
  world = [#world]

commit @browser
  world <- [#div style: [user-select: "none" -webkit-user-select: "none" -
moz-user-select: "none"]  children:
    [#svg #game-window viewBox: "10 0 80 100", width: 480 children:
      [#rect x: 0 y: 0 width: 100 height: 53 fill: "rgb(112, 197, 206)" sort:
```

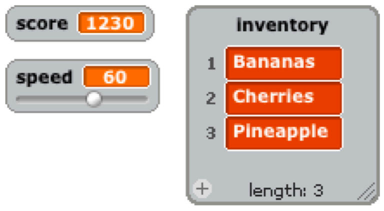
Variable monitor

Some systems allow probes to be installed on variables and expressions. Sometimes their values can be manipulated. This can be seen as a weak version of the technique I call *world as data;* attaching probes to parts of an opaque computational system means that parts of it become world-like—directly inspectable and manipulable.



```
function addone(x){
    return x + 1
}

addone(addone(addone(addone(1))))
5          4          3          2
```

```
5
```

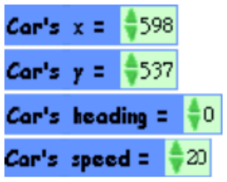


score 1230

speed 60

inventory	
1	Bananas
2	Cherries
3	Pineapple

length: 3

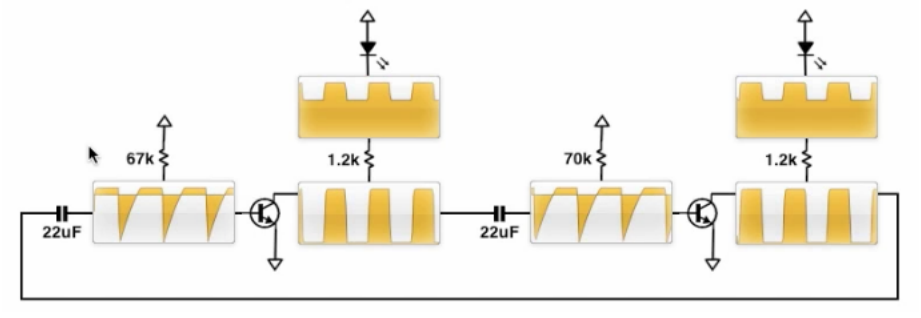


Car's x = 598

Car's y = 537

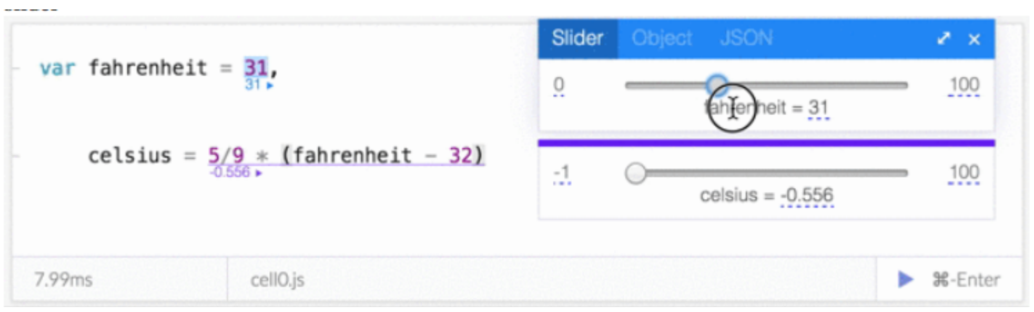
Car's heading = 0

Car's speed = 20



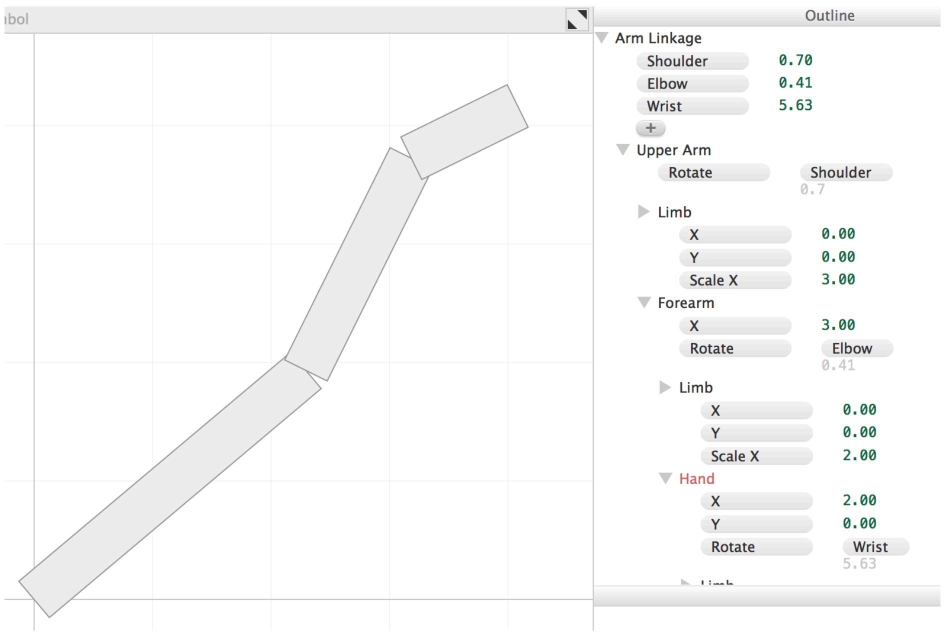
Backpropagation

The output of some systems can be directly manipulated, leading to indirect changes in input. Through back-propagation, input and output are coupled, establishing new ways for users to think and play with a system. This also enhances system tactility.



```
var fahrenheit = 31,
                 31 ▸

    celsius = 5/9 * (fahrenheit - 32)
              -0.556 ▸
```

```
0        fahrenheit = 31        100
-1       celsius = -0.556       100
```



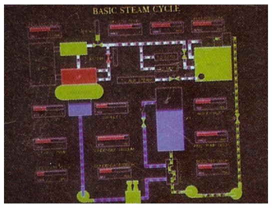
```
Outline
▼ Arm Linkage
    Shoulder        0.70
    Elbow           0.41
    Wrist           5.63
    +
  ▼ Upper Arm
      Rotate              Shoulder
                          0.7
    ▶ Limb
        X               0.00
        Y               0.00
        Scale X         3.00
    ▼ Forearm
        X               3.00
        Rotate              Elbow
                            0.41
      ▶ Limb
          X             0.00
          Y             0.00
          Scale X       2.00
      ▼ Hand
          X             2.00
          Y             0.00
          Rotate            Wrist
                            5.63
```

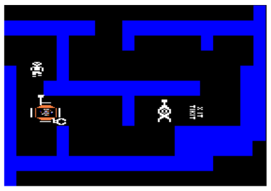
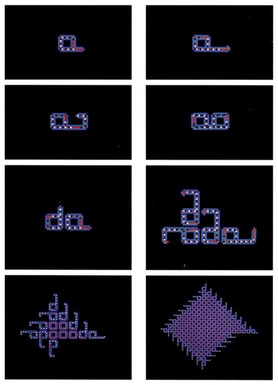
Tactile

Vivid tangible worlds turn abstract ideas into concrete and familiar things.

Live

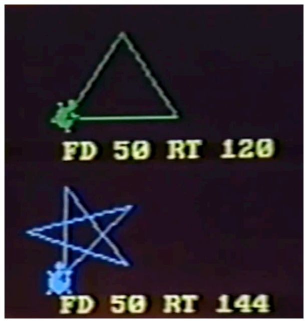
Aliveness means things are "always active and reactive" (Maloney and Smith 1995).



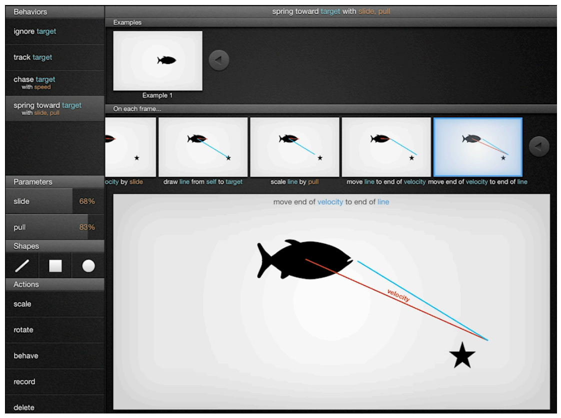
Syntonic

Mapping our bodies, egos, or cultures into a new domain helps us to learn and assimilate it.



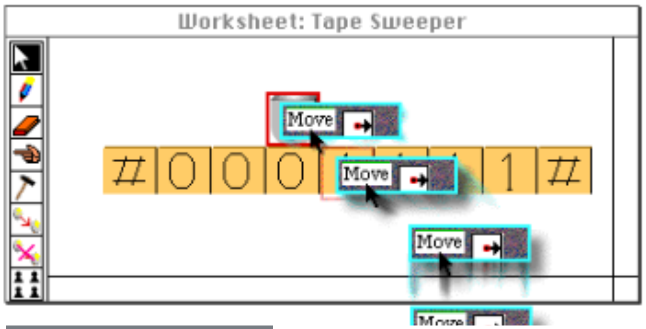
Concrete

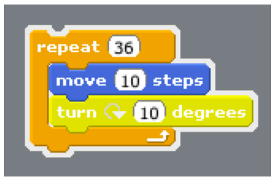
Example state is always present. Abstractions like procedures and variables are always accompanied by a *for example.*


Behaviors
ignore target
track target
chase target with speed
spring toward target with slide, pull
spring toward target with slide, pull
Examples
Example 1
On each frame...
ocity by slide
draw line from self to target
scale line by pull
move line to end of velocity
move end of velocity to end of line
Parameters
slide 68%
pull 83%
Shapes
Actions
scale
rotate
behave
record
delete
move end of velocity to end of line
velocity

Manipulate Things and Actions

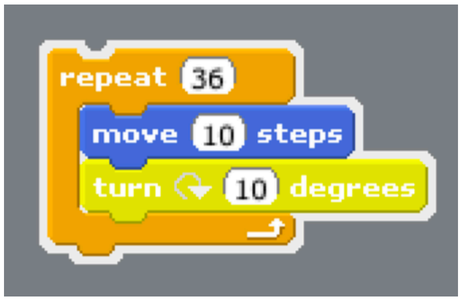
It is important that players can directly manipulate not just concrete objects, but concretized actions.



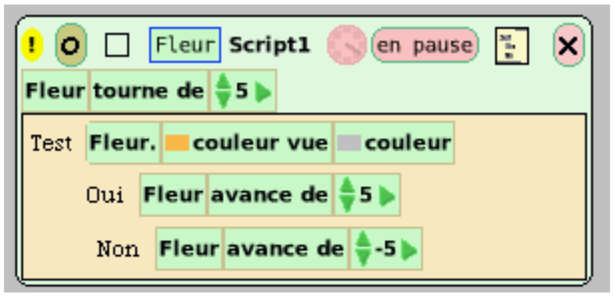


Tactile Code and Action

Procedures and actions can be made concrete.

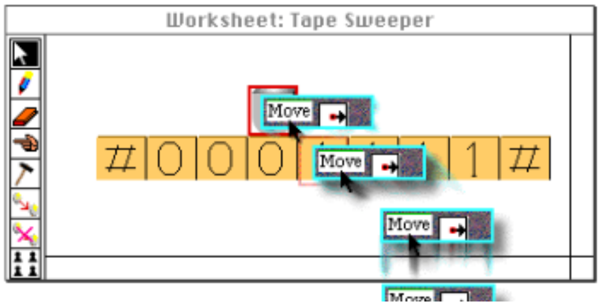


```
repeat 36
    move 10 steps
    turn ↻ 10 degrees
```

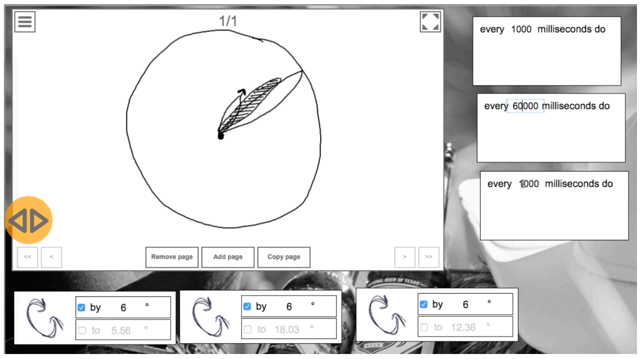


```
! O □ Fleur Script1 en pause ×
Fleur tourne de 5
Test Fleur. couleur vue couleur
    Oui Fleur avance de 5
    Non Fleur avance de -5
```

Block languages like Scratch and Etoys represent imperative language grammar and elements into tactile objects. Blocks can be dragged and snapped together, can be clicked to run, and visually display run-time status like errors and execution. This is an aid to comprehension and sharing (Repenning and Ambach 1996). Visual AgenTalk also allows blocks to be dropped onto objects:

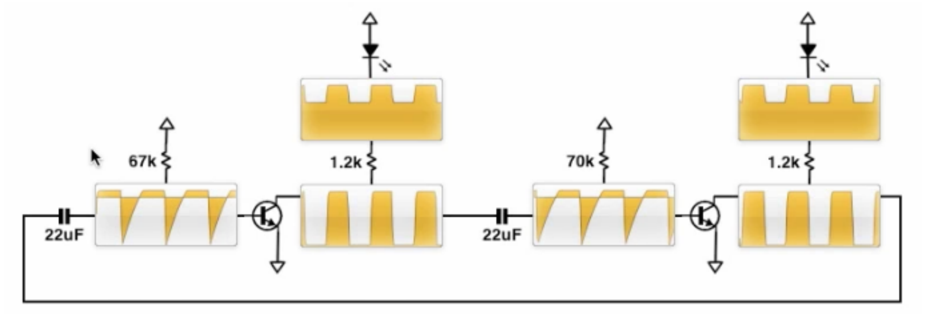


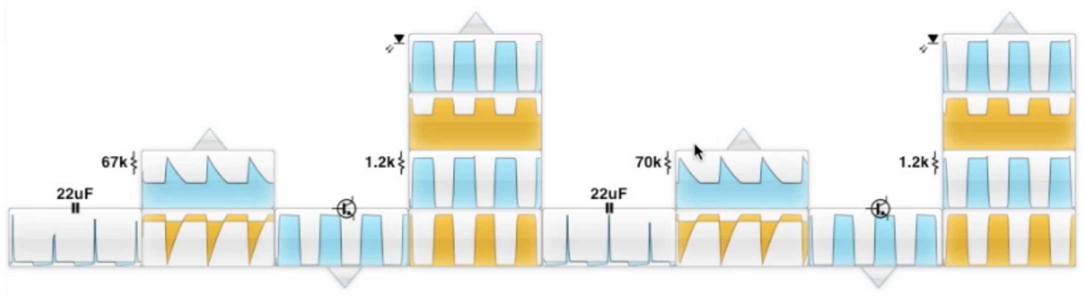
By recording the user’s manipulation of the world, tangible code elements can also be *composed by example* (Repenning and Ambach 1996), as shown below in this example from Lively Kernel:


1/1
every 1000 milliseconds do
every 60000 milliseconds do
every 1000 milliseconds do
Remove page
Add page
Copy page
by 6 °
to 5.56 °
by 6 °
to 18.03 °
by 6 °
to 12.36 °

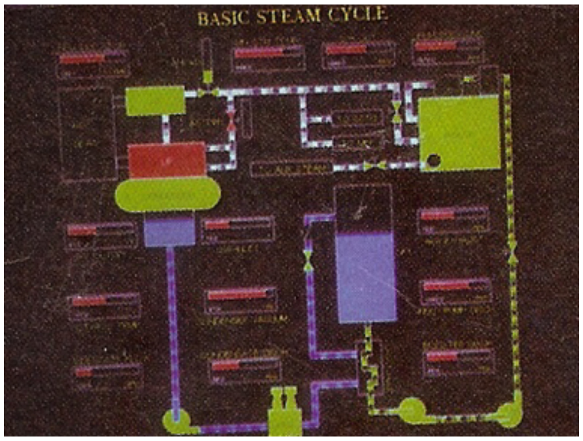
See data flow

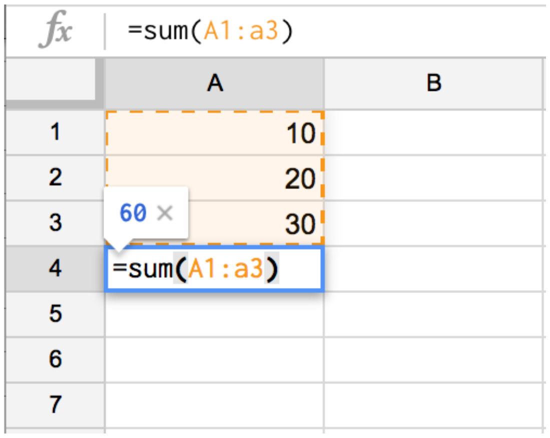
When programs are described via diagrams and state data is concretely shown this creates the effect of being able to see data flow through the system. In these images from Bret Victor's *Media for Thinking the Unthinkable (2013)*, a circuit diagram is annotated with wire voltages, and then the components are replaced with plots of current:





Steamer is an "interactive inspectable simulation" of a steam ship (Hollan et al. 1984). It is a dynamic and hierarchical "dynamic graphical explanation" of a non-trivial domain—the propulsion system of a Navy ship, modeled in about 100 different diagrams. (It also includes an authoring tool for the diagrams.) Spreadsheets function similarly, exposing intermediate computations and values.





fx =sum(A1:a3)

	A	B
1	10	
2	20	
3	30	
4	=sum(A1:a3)	
5		
6		
7		

60 ×

Personally Meaningful

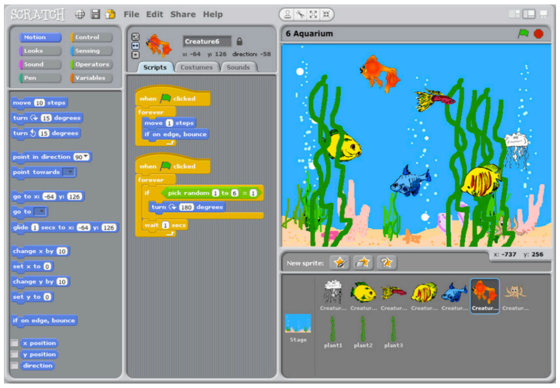
You should be able to do something you care about.

Personalizable

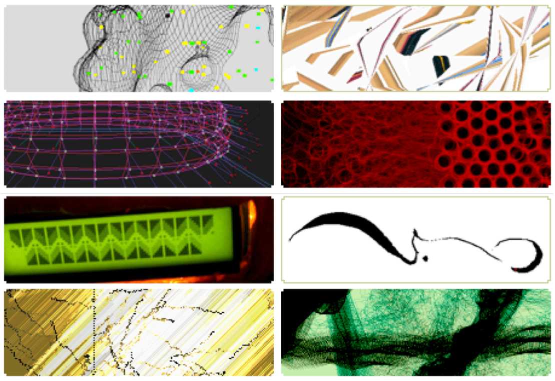
It's important that users make and engage with things they authentically care about.

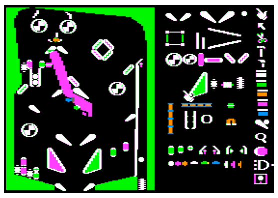
This is a reason why art- and world-making, as in Logo, Scratch, Processing, and Minecraft, are such pervasive and powerful themes—they offer a direct avenue towards personally meaningful activity. (Constructionism harnesses the appeal of project-making towards learning outcomes.)

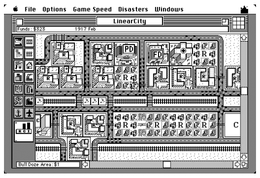
Open ended materials can lend themselves to appropriation—fulfilling the novel goals and purposes of players.



```
when [flag] clicked
forever
    move (1) steps
    if on edge, bounce

when [flag] clicked
forever
    if <(pick random (1) to (6)) = (1)>
        turn (180) degrees
    wait (1) secs
```





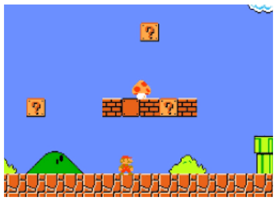


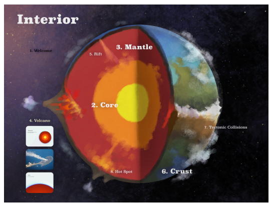
Directed and Undirected

A balance must be found between structure that guides and opportunities for self-direction.

Gentle On-ramping

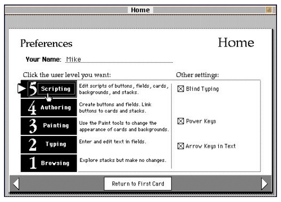
New players must be made to feel welcome, and guided towards expertise.





Progressive Disclosure

Gradually introducing new features affords players an opportunity to assimilate them, and allows designers to thoughtfully stage their introduction. (e.g. stories, games, authorship levels, and user frames.)



Home

Preferences

Home

Your Name: Mike

Click the user level you want:

5 Scripting	Edit scripts of buttons, fields, cards, backgrounds, and stacks.
4 Authoring	Create buttons and fields. Link buttons to cards and stacks.
3 Painting	Use the Paint tools to change the appearance of cards and backgrounds.
2 Typing	Enter and edit text in fields.
1 Browsing	Explore stacks but make no changes.

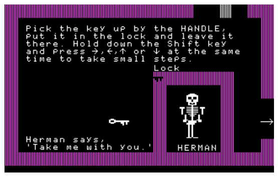
Other settings:

☒ Blind Typing

☒ Power Keys

☒ Arrow Keys in Text

Return to First Card

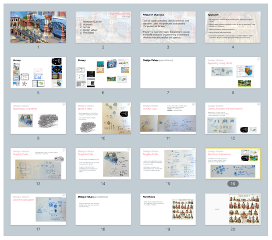


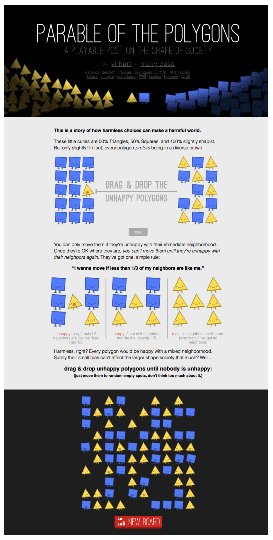
Pick the key up by the HANDLE, put it in the lock and leave it there. Hold down the Shift key and press →,←,↑ or ↓ at the same time to take small steps.

Herman says, 'Take me with you.'

Structures

Experience can be shaped via journeys—obstacle filled stories—and containers like slide decks, reports, posters, and journals.





PARABLE OF THE POLYGONS

A PLAYABLE POST ON THE SHAPE OF SOCIETY

by vi hart + nicky case

This is a story of how harmless choices can make a harmful world.

These little cuties are 50% Triangles, 50% Squares, and 100% slightly shapist. But only slightly! In fact, every polygon *prefers* being in a diverse crowd:

You can only move them if they're unhappy with their immediate neighborhood. Once they're OK where they are, *you can't move them until they're unhappy with their neighbors again.* They've got one, simple rule:

"I wanna move if less than 1/3 of my neighbors are like me."

unhappy: only 1 out of 6 neighbors are like me. less than 1/3.

happy: 2 out of 6 neighbors are like me. exactly 1/3.

meh: all neighbors are like me. (also meh if i've got no neighbors)

Harmless, right? Every polygon would be happy with a mixed neighborhood. Surely their small bias can't affect the larger shape society that much? Well...

drag & drop unhappy polygons until nobody is unhappy:

(just move them to random empty spots. don't think too much about it.)

Sandbox

Loose parts afford open-ended play, improvisation, experimentation, and reinterpretation. In some systems even the explanatory materials can be disassembled. Robustness fosters feelings of trust and safety.