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.
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 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 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 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 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
Appealing and computational worlds draw people in.

**Appealing**

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

**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).

**Explore, Build, and Inhabit**

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

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:
Zoom and Encapsulation

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

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.

Schematic
As in a toy or diagram, simplicity and transparency make representations clear and easy to absorb.
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 `Player`, a `BiPlayer`, and some `PlayerControls`. 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 affect:

- The `PlayerPosition` tracks the distance the player has traveled, the current game screen, and the high score.
- The `BiPlayer` stores his current position and (vertical) velocity.
- The `PlayerControls` 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 position of its gap, and continue on.

Setup

Add a `Flappy` and a world for it to flap in:

```javascript
setup

// Player controls
PlayerControls new = new PlayerControls();

// Game world
GameWorld new = new GameWorld();
```

Next we draw the backdrops of the world. The `Player` and `Obstacle` will be drawn later based on their current states. Throughout the app we use resources from Bhaskarun’s Flappy bird demo in Clojure. Since some of these things change over time, we commit them once when the player starts the game.

Draw the game world:

```javascript
draw

// Player
Player new = new Player();
```

Now we can see if the player’s `PlayerPosition` is going to run into the `Obstacle`:

```javascript
if (new(PlayerPosition).x * 2 > new(Obstacle).width + new(PlayerControls).getOffset()) {
    // Game over
    break;
}
```
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.

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

```javascript
addone(addone(addone(addone(1))))
```

Score: 1230

Speed: 60

Inventory:
- 1: Bananas
- 2: Cherries
- 3: Pineapple

Car's x = 598
Car's y = 537
Car's heading = 0
Car's speed = 23
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.
**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.

**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.

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:
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.
Appealing

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.

Personalizable

You should be able to do something you care about.

Personally Meaningful
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.)

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

**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.