DrawTalking:
Building Interactive Worlds by Sketching and Speaking

Karl Toby Rosenberg
New York University
New York, New York, USA
ktr254@nyu.edu

Rubaiat Habib Kazi
Adobe Research
Seattle, Washington, USA
rhabib@adobe.com

Li-Yi Wei
Adobe Research
San Jose, California, USA
liyiwei@acm.org

Haijun Xia
University of California, San Diego
La Jolla, California, USA
haijunxia@ucsd.edu

Ken Perlin
New York University
New York, New York, USA
perlin@nyu.edu

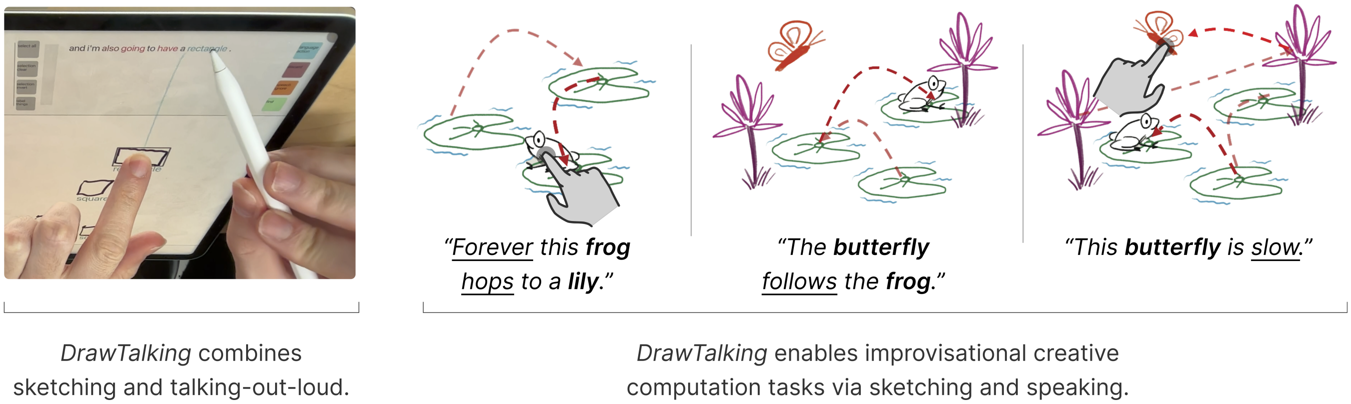

"Forever this **frog** hops to a **lily**."
"The **butterfly** follows the **frog**."
"This **butterfly** is slow."
DrawTalking combines sketching and talking-out-loud.
DrawTalking enables improvisational creative computation tasks via sketching and speaking.

Figure 1: Our approach *DrawTalking* mediates sketching and talking-out-loud through direct manipulation, enabling many use cases across improvisational creative tasks.

ABSTRACT

We introduce DrawTalking, an approach to building and controlling interactive worlds by sketching and speaking while telling stories. It emphasizes user control and flexibility, and gives programming-like capability without requiring code. An early open-ended study with our prototype shows that the mechanics resonate and are applicable to many creative-exploratory use cases, with the potential to inspire and inform research in future natural interfaces for creative exploration and authoring.

CCS CONCEPTS

• **Human-centered computing** → **Graphical user interfaces**; **Natural language interfaces**; **Interaction techniques**; **Interactive systems and tools**;

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676334

KEYWORDS

creativity, sketching, play, programmability, multimodal, human-AI collaboration, prototyping

ACM Reference Format:
Karl Toby Rosenberg, Rubaiat Habib Kazi, Li-Yi Wei, Haijun Xia, and Ken Perlin. 2024. DrawTalking: Building Interactive Worlds by Sketching and Speaking. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 25 pages. https://doi.org/10.1145/3654777.3676334

1 INTRODUCTION

Sketching while speaking aids innovation, thinking, and communication — with applications in animation, game design, education, engineering, rapid prototyping, storytelling, and many other creative and spontaneous activities [14, 65]. The combination enables us to think about and share anything through make-believe – including things that do not or cannot exist. We achieve this by assigning *representations* (sketches) to *semantic concepts* (objects, behaviors, properties) [64]. For example we might suspend our disbelief so a square represents anything such as a house, a playing card, a map, a dog, or a person.

Furthermore, we often engage with an audience or conversational partner while drawing and talking, whether it's through storytelling to one's child or explaining concepts during a whiteboard lecture. These types of drawing and talking interactions have a spontaneous, playful, exploratory, and improvisational feel with fluid interactions. Storytelling and drawing while interacting with a standard GUI is arguably challenging if we wish to achieve this feel and fluidity. The goal, then, is to explore drawing and talking for these scenarios as an alternative, additional input scheme.

Specifically, we explore how we can take advantage of the presence of narrative to support richer human-computer interactions in creative world-building, where speech can both be used as verbal storytelling addressed to people *and* as a simultaneous input to the machine. **We are motivated to design interactions built around this presence of narrative to enable richer human-computer interactions in creative world-building.**

This work is an attempt to realize a style of spontaneous interaction that seamlessly integrates sketching and talking-out-loud to build interactive, explorable worlds, thus greatly increasing our range of computational expression [66]. In the design of these interactions, we explore the use of speech and direct input in the context of storytelling. Prior works in interactive sketching [18, 49, 53, 61, 63], language/AI-mediated interfaces [7, 16, 24, 32, 58, 68–70], and visual programming-adjacent interfaces or games [10, 13, 23, 31, 48, 50, 62] have laid valuable groundwork. However, they often require content to be pre-built or assume that users have predetermined goals, limiting the user's potential to be spontaneous. Tools focus on generating a specific output rather than facilitating an ongoing creative process [8]. They might enforce given representations of objects (e.g. realistic or specific sketch recognition). They might feature complex UI, and in the case of programming-oriented tools, require explicit programming knowledge (whether via text, nodes, or blocks).

In our prototype system, *DrawTalking*, users speak while freehand-sketching to create, control, and iterate on interactive visual mechanisms, simulations, and animations, with the freedom to tell stories using the same speech input as part of the narrative[1]. Via speech and direct manipulation, the user names their sketches to provide semantic information to the system, and narrates desired behaviors and rules to influence the interactive simulation, as if by explaining to others or telling a story. This is inspired by the way in which people might speak and/or write text to describe objects as they are being drawn.

The core aspect of the user-machine interaction design is the simultaneous use of speech as narrative, and as input to the system, to enable fluid storytelling and building of interactive worlds. The user narrates, points, and draws as if telling the story of their drawings to an audience. The user's narration communicates intent to the machine by labeling (naming) or commanding (directing) objects. As a result, the user gains fluid programming-like capability *through* their narration, as opposed to needing to narrate while multitasking with a complex GUI and the potential audience. This allows for a natural extension to the GUI in the context of storytelling.

The second key insight is that narration and pointing as in storytelling can provide the machine with names and properties belonging to sketches, so in this storytelling context, the machine does not need to guess the user's intended meaning behind their drawings. As a result, users can build and control interactive worlds by simple combinations of sketching and speaking while telling a story.

DrawTalking, in addition, requires no preparation or staging step, and supports the user in making changes to content and behavior anytime. By design, we balance AI-automation with user-direction such that the user is the one who chooses how to represent content, when to do operations, and what logic to define.

In sum, we contribute:

- *DrawTalking*: a novel sketching+speaking interaction with a design supporting fluid worldbuilding within the context of storytelling. It aims to balance user-agency with machine automation.
- *A demonstration of DrawTalking as a multi-touch prototype application for the iPad.*
- *An early qualitative study of DrawTalking that reveals its use for emergent creativity and playful ideation.*

2 RELATED WORK

Our research is based on a confluence of advances in multimodal, sketching, and programming interfaces.

2.1 Natural Language-Adjacent Interfaces

Systems such as SHRDLU [68] and Put That There [7] pioneered the vision of employing natural language to communicate with computers. Due to recent advances in speech recognition and natural language understanding, the popularity of this interaction modality has exploded, and has been used in a wide range of domains. For example, VoiceCut [22] and PixelTone [24] allow users to speak short phrases or sentences to perform desired operations in image editing applications, but these applications are heavily domain-specific. Tools like WordsEye [9], Scones [16], and CrossPower [69] enable scene generation or content editing via language, and interfaces such as Visual Captions [32], RealityTalk [29], and CrossTalk [70] use language structure to make content appear during talks or conversations. However, many of these interfaces tend to assume that the user knows what they want to create in advance — i.e. an end-product with an initial goal. They require an initial phase in which the user must define content and behavior up-front. This could limit open-ended exploration during the creative process, when the user does not necessarily have an end-goal in mind. Further, the majority of such interfaces use language input to generate or spawn content without the user in the loop. An alternative is to enable greater interactive control and definition of the behavior of the content. Our approach emphasizes flexibility and user-control during the creative process. The user can define and iterate on content at any point. Our prototype specifically supports definition of behaviors within an interactive simulation. Within this prototype, we explore language input, combined with direct manipulation, as a way of empowering the user to program and control scenes interactively. Notably, although speech+pointing commands aren't new (e.g. "Put That There" [7]), we focus on the use of speech input as leveraging a narrative that is required *even when* using standard input.

[1] Informally, we distinguish between storytelling and narration: storytelling is a process achieved by mixing possibly many modalities (narration, speech, text, visuals). Narration refers to the use of speech or text — possibly, but not always for storytelling.

2.2 Dynamic Sketching Interfaces

HCI researchers have extensively explored sketching interfaces for dynamic and interactive visualizations ever since the first graphical user interface (GUI) SketchPad [61] and William Sutherland's thesis, the forerunner of the visual programming language [62]. Many works use direct manipulation and sketching techniques to help users craft interactive behaviors and toolsets for illustrated animation, UI, and visual-oriented programs. For example, works by Kazi et al. [18, 19], Landay et al. [23], Saquib et al. [53], and Jacobs et al. [17] focused on mixing illustration, programming, and prototyping. Programming by-demonstration is featured in works such as K-Sketch [11] and Rapido [27]. Scratch [50] is a well-known visual programming environment mixing game-like interactions with user-provided content and images for a playful experience. texSketch [59] supports the user in forming connections between texts and concepts to learn via active diagramming. Our interface is framed in a complementary way around correspondences between sketches and language elements, but we use sketch-language mapping as a control mechanism, enabling the user to create interactive simulations and behaviors for open-ended exploration.

Prior work also explored supporting development of pre-programmed simulations and domain specific behaviors to craft interactive diagrams. In Chalktalk, for example, the system uses sketch recognition to map a user's hand drawn sketches into corresponding dynamic, pre-programmed behaviors/visualizations [49]. More domain-specific tools like MathPad2 [25], Eddie [54], PhysInk [55] and SketchStory [26] use hand-drawn sketches and direct manipulation interactions to create interactive simulations in physics, math, and data visualization.

2.3 Programming-Like Interfaces

The customizability and flexibility of such a more general interface implies a need for programmability. Examples of these include real-time world simulation systems and programmable environments such as the SmallTalk programming language[15], Scratch [34, 50] , Improv [48], ChalkTalk[49], and creative world-building games like the Little Big Planet series [30, 31, 52] and Dreams [13]. These encourage interactive building of scenes, games and stories. They combine elements of interactive visual programming and drawing/sculpting with many types of content (2D, 2.5D, 3D, images). However, all use explicit interfaces for programming or programming-like functionality (nodes, wires, text). For example, voice-enabled AI interfaces like StoryCoder [12] integrate a collaborative agent with which to communicate to create story sequences, but still use the block-based programming GUI as in e.g. Scratch. PUMICE [28] is an AI agent specializing in defining new tasks on existing mobile GUIs via a conversation with the user. To learn unknown functionality, the agent repeatedly asks the user to demonstrate by direct manipulation with the app or to explain it by speech. Although comparable to our approach in terms of using speech+direct manipulation as a form of programming, PUMICE requires interaction with a chat-bot-like interface by design, and requires existing target applications (a different, specialized use case as opposed to ours). In contrast, we wanted to support simultaneous narration and (potentially) interaction with an audience while building a scene, with no pre-existing application or goal in-place. This leads to different interaction design criteria: we choose not to use an AI assistant because this could divert attention from an audience. Rather, the user speaks in narrative form as in storytelling and mixes direct manipulation (touch, drawing). The same input functions as both storytelling to an audience and commands to the machine, so the user does not need to address an assistant.

Overall, we depart from explicit UI for sketching+programming-like capability, and largely replace much programming-like functionality with the use of language. Our direction explores the use of verbal, descriptive story narration together with other input modalities (i.e., touch and pen input) to create animated and interactive graphics through sketching while potentially communicating to an audience as one might during teaching or storytelling.

3 FORMATIVE STEPS

We derive a set of design goals by examining existing practices. This would also inform the development of interaction techniques and a prototype interface.

3.1 Methodology

We used a mixed-method approach for our study consisting of an analysis of online instructional videos and conducting a set of sketching design sessions.

To get a sense of the relationships between speech, text, and drawings, and to find examples of content that people created, we conducted an informal (non-exhaustive) search for example content online. We looked for examples that involved creating (or showing) rough sketches while speaking, e.g. from popular video channels and educational course recordings. (Refer to appendix Figure 14 for additional sample materials.)

To gain further insights about workflow and technical requirements, we conducted informal exercises with 6 participants $P1_{init}$-$P6_{init}$ to observe casual sketching and speaking process with little to no preparation. All participants had some experience with sketching-out ideas (e.g. concepts, storyboarding, project/presentation sketching for game design). Each participant was asked to think of (at least) 1 personal topic they would feel comfortable narrating while drawing with freehand sketches. Participants used their choice of tools during the session (eg, MS Paint, Notability, basic tools in Photoshop), but we only allowed basic color selection and transformations to keep tool usage roughly the same. (Refer to appendix Figure 15 for P1's result.)

3.2 Results and Observations

3.2.1 Association between spoken language, text, and drawing: Participants used a variety of visual styles (abstract or cartoon), symbols, and diagrammatic elements to express their ideas. They would sometimes label their objects with names or use in-line text to describe elements in the screen for them to reference later [2]. Pointing or proximity to sketches while using deictics also acts as a way to refer to objects (as observed in content recorded with cursors or speakers' hands.) In short, semantics are associated with sketches in many ways, including through narration, or explicit text labels. *The natural association between spoken language, text, and sketches inspired our our main interaction technique.*

3.2.2 Temporal synchronization between speech and drawing: In the content search and exercises, people do not tend to synchronize their speech exactly when drawing specific objects (or if they do, the delay between actions is unreliable). However the order in which people mention sketches verbally usually will correspond to pointing and sketching actions[47]. *This means we can use this ordering property with respect to speech to let the user map semantics to objects in sequence, but temporal synchronization is unreliable.*

3.2.3 Drawn versus non-drawn spoken content: Although narration was used to explain the content on-screen, not all content (such as additional animations in the case of produced-content or additional details) and not all narration mapped to each other. In other words, there is content that the user doesn't necessarily describe via language and some content might be considered unnecessary to represent. *Hence, the modalities complement each other to represent the complete story.*

3.2.4 Speed and flow: People operate at many different speeds when drawing and talking, and unpredictably move back and forth between sketches as they're iterated on. This means that *we can't force people to use a specific speed or cadence.*

Additionally, participants would sometimes pause to locate nested UI components (e.g. opening a color-picking menu) or switch between pen and eraser modes. *To complement the GUI, speech could allow users access functionality by referring to it without needing to find it. This could help users keep the flow of their narration.*

3.2.5 Content modification: For the exercises, a participant would usually multitask and refine their sketches over time after initially mentioning the entity. For example, $P1_{init}$ added colors to their gray bird and pond sketches as they described their experience at the scene. Additionally, the participant might make verbal corrections (e.g. $P1_{init}$ initially called a "pond" a "lake," but self-corrected verbally). This means that speech errors are natural in real-life and we ought *to provide ways to correct it* [60]. *As users' intentions change, users should be able to update their content.*

3.2.6 Hierarchical object model: When referring to objects on-screen, narrators would name objects, refer to existing objects to draw attention to them, and define hierarchies and relationships between objects. In short, sketched presentations and content implicitly encoded a *hierarchical object model, describing entities and the relationships between them, as well as ways to refer back to them.*

Based on our formative observations, we needed to prioritize user control over a variety of possible creative workflows. Rather than a fully-automatic solution, we needed a synthesis of user-directed input and system-feedback in support of spontaneous creative processes.

3.3 Design Goals

We envision an interface for creative exploration with interactive capability that (a) is controlled via drawing and talking in the context of narration, (b) does not impose many assumptions about the user's intent or content, (c) focuses on the process, not just on the artifact, and (d) does not require programming knowledge. Above all, the user should have control.

To that end, we wanted an interface that:

D1 *makes minimal system assumptions*
where the user controls the creative process, and the representation and behavior of sketches.

D2 *is flexible, mutable, fluid, and playful*
in that it supports improvisation, quick changes, and rapid iteration of ideas with a spontaneous feel, where operations are easily accessible and doable in any order.

D3 *is transparent and error tolerant*
by telegraphing what the system's understanding is and providing multiple opportunities for users to make changes or recover from system or user error.

D4 *supports programming-like capability*
without the need for a coding interface.

4 DRAWTALKING

We designed and developed DrawTalking, an approach to creating interactive worlds using freehand-sketching and speech. It is based on design goals derived from our formative study and implemented as a pen+multi-touch application.

4.1 Concept

In DrawTalking, speaking serves a dual-purpose, enabling simultaneous narration and interaction with the interface for storytelling and world-building: the user can explain concepts and tell stories, and at the same time draw, refer to, and label the objects in their sketched world with semantics (e.g. nouns, adjectives, adverbs) determining objects' names and behaviors.

The *user-specified* labeling tells the system what the objects are, irrespective of their visual representation, making them controllable via the narration.

The user can furthermore create the rules to automate behavior between objects in the simulation. Touch controls also allow the user to interact directly with the simulated world. This results in a free-form sandbox for animation and programming-like behavior that mixes direct user-control with machine automation. An overall workflow of DrawTalking in shown in Figure 3.

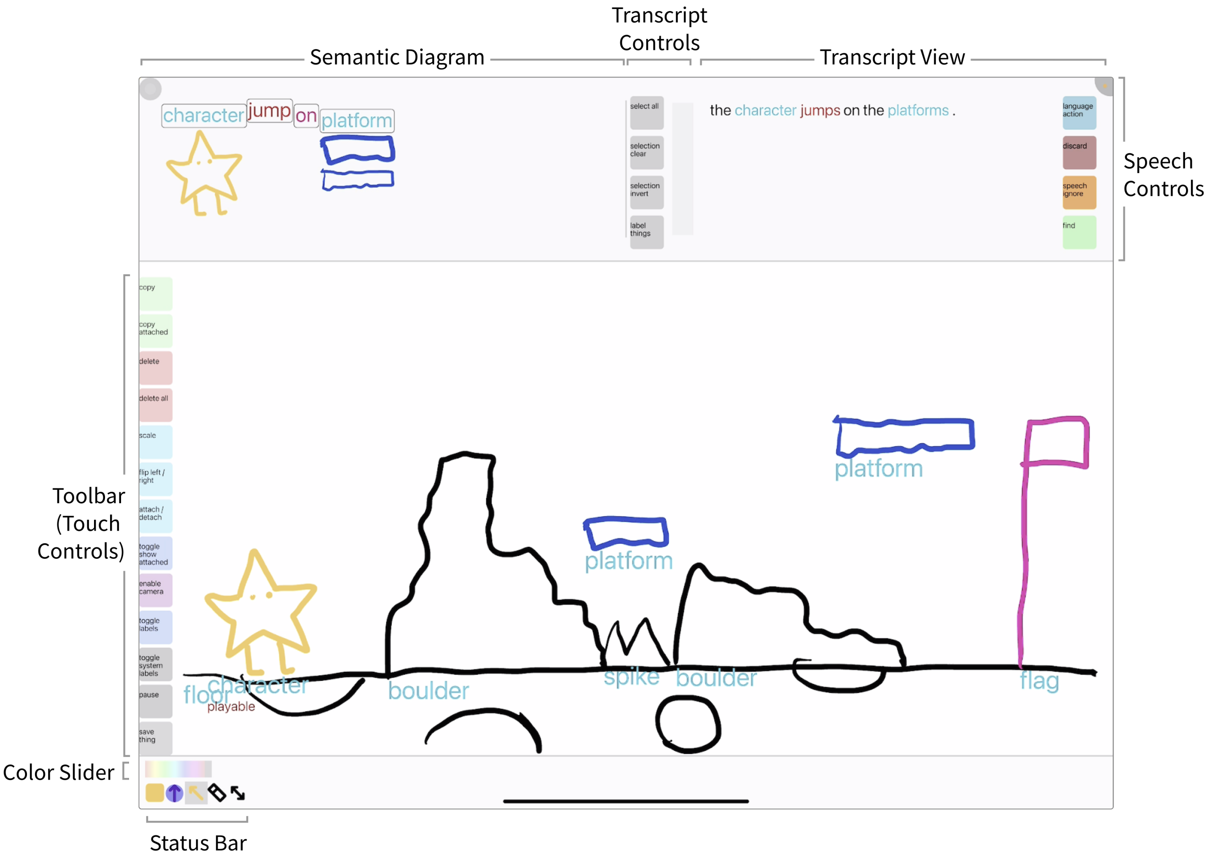
4.2 User Interface Components

The interface (Figure 2) exposes a *transcript view* (Figure 2) and semantics diagram (Figure 4) to make system understanding of input transparent, quickly editable and accessible, and error-robust, as per our interface design goals (subsection 3.3).

Speech recognition is continuous for interactive use, so the *transcript view* lets the user visualize and optionally edit speech input (see Figure 5), assign names and attributes to sketches using the pen linking modality, and stage/confirm commands.

When ready, the user taps "language action" to stage a command. The diagram appears and displays a visual for the machine's understanding of the input. It provides a way to reassign objects within the command (if the user wants). The user confirms with the same button to execute the command, or cancels with the discard button.

Complementary to the sketching and language workflows, the find panel along with the *transcript view* enables the user to find and teleport to objects by semantic search, which helps avoid losing track of objects. (This use of speech is inspired by real-world interaction, in which we use language to talk about real, distant,



the character jumps on the platforms .

Figure 2: Interface Overview: An interface screenshot (from P4 in section 6). The toolbar (left) enables edit operations, e.g. copy, delete, attach/detach, save sketch. The transcript view displays the user's speech input in an interactive panel, and the semantic diagram displays the machine's understanding of the input. "Speech controls" stage/confirm an action, discard input, and toggle speech recognition. "Transcript controls" offer quick transcript text selection. The status bar displays the current color, a compass pointing "up," and has the pen/eraser state-change buttons and indicators. The scene shown is just before the user confirms a command for the utterance *"The character jumps on the platforms"*; it selects the sketch labeled "character" and all sketches labeled "platform." Tapping the "language action" button (top-right) will stage the command and display the semantic diagram (top-left) representing the machine's understanding of the input with selected objects displayed under their respective words; the user confirms the command by tapping again. This causes the character to jump on all of the platforms. For details on the workflow and commands, see subsection 4.2 and subsection 4.4.

invisible, imaginary, or conceptual objects.) Additionally, it enables a means to view, toggle, or delete the current commands and rules acting on the world simulation (See Figure 6).

In sum, all three of these sketching-language interface components combine to support the user's process via our design goals. Together, they provide transparency into the system's state, multiple forms of editability and error-tolerance, and multiple optional levels for user control.

4.3 Sketching and Labeling

Sketches are independently-movable freehand drawings, text, or numbers created by the user. The user labels their sketches to make them controllable: nouns (names) for unique identification and adjectives and adverbs (properties) for modulating sketch behavior. For flexibility, we offer two direct ways to label (or unlabel).

(1) tap 1 to many sketches and speak with deixis [57] (e.g. *"this/that is a <noun>"*, *"this/those are <noun>s"*).
(2) touch sketches + pen-tap words to link at any time with any text in the history, as opposed to only the current speech, enabling freer narration.

Labeling works in tandem with the user interface elements (4.2). See Figure 3 for an example workflow.

4.4 Language Commands and Functionality

DrawTalking interprets the structure of language input into commands built from primitives. These primitives are intended to be a small sample of possible functionality that demonstrate our working concept. (They are partly inspired by existing software and

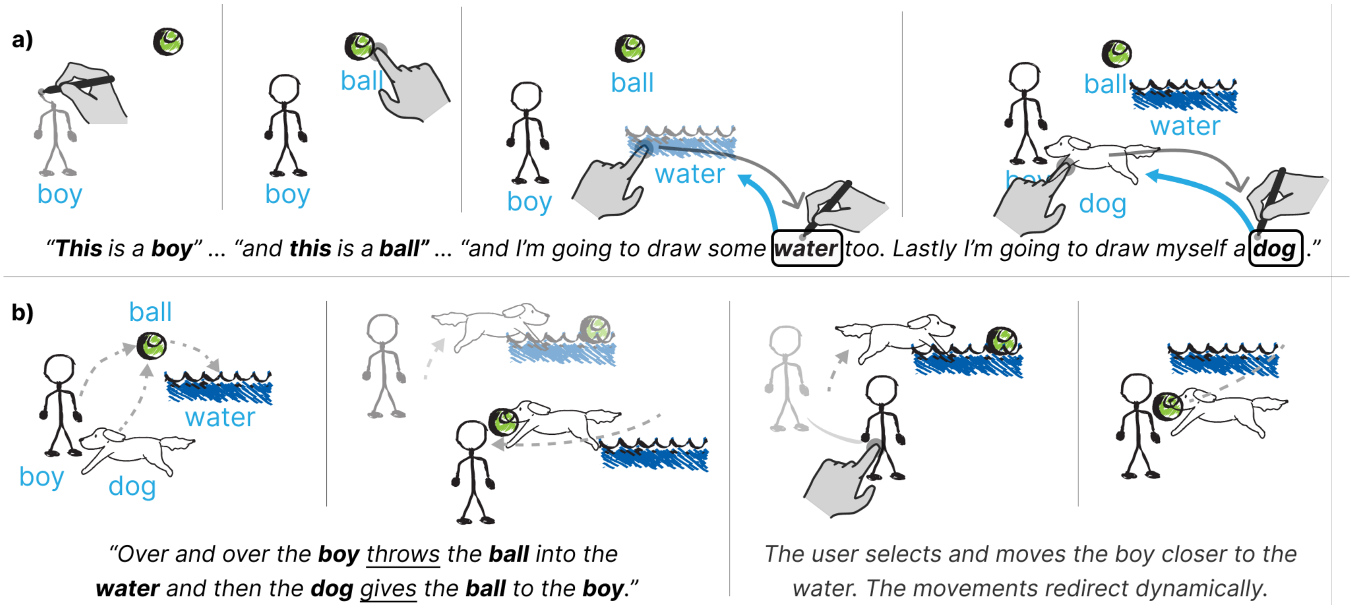

a)
boy
boy
ball
boy
ball
water
ball
water
dog
"**This** is a **boy**" ... "and **this** is a **ball**" ... "and I'm going to draw some **water** too. Lastly I'm going to draw myself a **dog**."
b)
ball
water
boy
dog
"Over and over the **boy** throws the **ball** into the **water** and then the **dog** gives the **ball** to the **boy**."
The user selects and moves the boy closer to the water. The movements redirect dynamically.

Figure 3: Overall workflow: Dog and boy's infinite game of fetch.
a) **From left to right, the user draws and labels them using multiple approaches at different stages of drawing. a.1) The user is midway through drawing the boy, but can label it using "This is a boy" as the pen is interacting with the object. a.2) The ball is already drawn, but unlabeled; during the same sentence the user can quickly tap the ball to label it. a.3) The user draws some water and uses free-form speech to say "water" without deixis (such as "this"/"that"). The user can select the object with touch and simultaneously tap the word "water" in the transcript label the object with the word in the transcript. a.4) Touch+pen on a word will remove the label. Adjectives work in the same way.**
b) Left: **Labeled sketches are commanded.** ***Right:*** **Interactive user-participation. The user can move objects as the system is simulating their movements, which will dynamically adjust as the user plays-around spontaneously.**

design spaces [13, 50, 56].) (See the Info Sheet supplemental material.) We emphasize that our contribution is the interactive *way* of controlling elements. Our specific application and the fidelity and comprehensiveness of the supported behaviors are implementation details meant as a minimal demonstration of the DrawTalking *concept*. In other words, animation complexity and visual fidelity aren't limitations inherent to the interface concept. These can be improved by implementers without rethinking the interaction design or contribution. We focus on DrawTalking's expressiveness as a control mechanism for any potential behaviors.

Verbs are actions performable by sketches and the system, either built-in or *user-defined in terms of other verbs by composition of existing primitives*. Examples of implemented verb primitives include:

- animations (e.g. *move*, *follow*, *rotate*, *jump*, *flee*)
- state changes (e.g. *create*, *transform*)
- events (e.g. *collide with*, *press*)
- inequalities (e.g. *equal*, *exceed*)

Verb behavior changes based on other parts of the sentence:

Conjunctions run simultaneously (e.g. "The dog jumps *"and"* the cats jump").
Sequences run in-order (e.g. "The dog jumps *"and then"* the cats jump).
Stop commands cancel an ongoing operation (e.g. "The square stops moving").
Prepositions (like *on*, *to*, *under*) cause verbs to exhibit different behavior, e.g. *"The dog jumps on/under the bed"* impacts the dog's final position relative to the target. All verbs can use any such prepositions as input. Prepositions also might describe the spatial relationships between objects in a command (Figure 7).
Timers specify the duration of a verb, e.g. *"the square moves up for 11.18 seconds and then jumps."*.
Loops repeat an action, either forever (e.g. *"endlessly the dog jumps"*), or finitely (e.g. *"10 times the dog jumps excitedly"*).

Special verbs include *"become"*, which modifies the sketch's labels, and *"transform into"*, which also instantly replaces a sketch's visual representation with another's to support state changes, e.g. *"the sun transforms into a moon"* or *"the frog transforms into a prince"*. "Follow" also offers a simple specialization on nouns — if a system object "view" is commanded to follow a given object, the system camera will track that object. e.g. *"The view follows the hero."*

Nouns, pronouns, and deixis refer to object labels and are used to pick *specific* objects or specify *types* of objects. Using deixis while selecting an object will select the object immediately; pronouns can refer-back to objects, enabling commands that read less repetitively and more naturally: (e.g. "The dog jumped. *She* jumped again.")

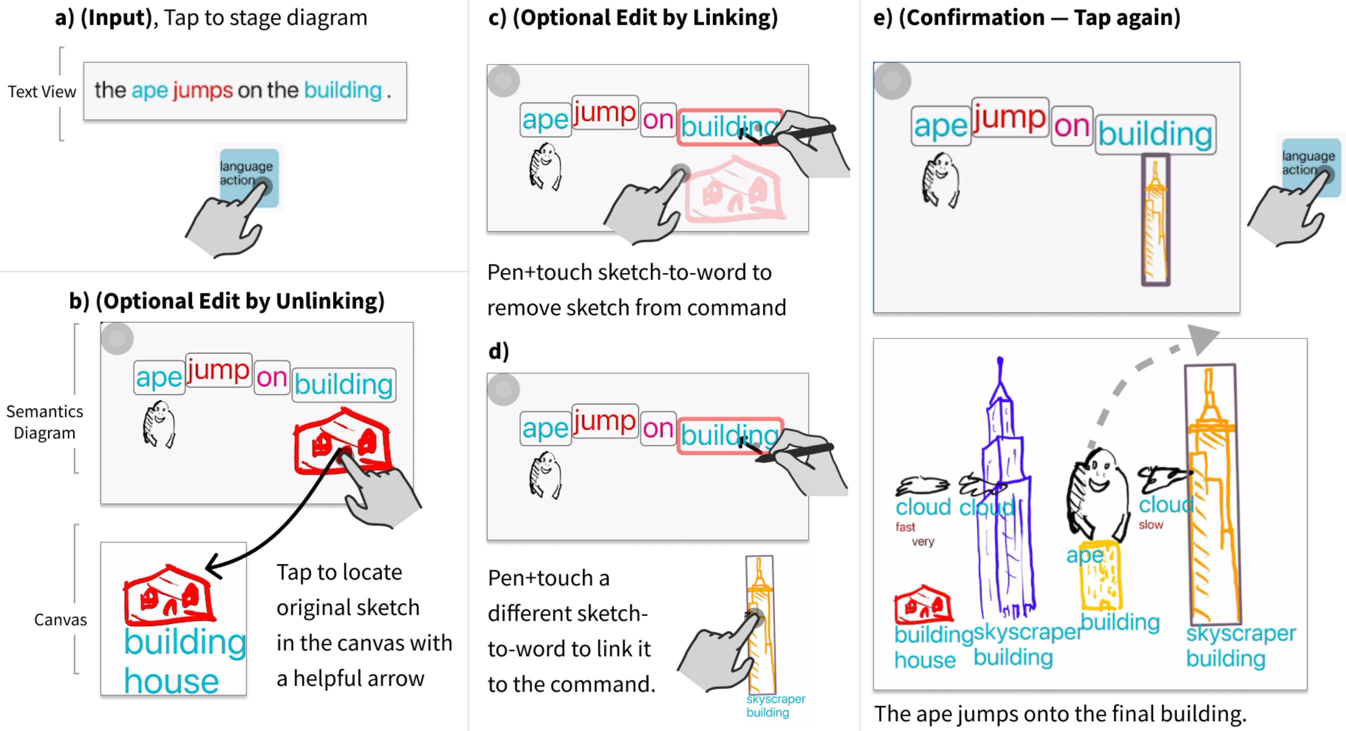
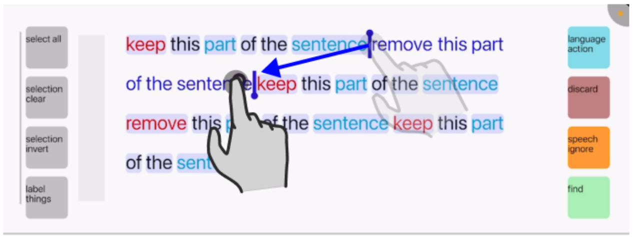

a) (Input), Tap to stage diagram
Text View
the ape jumps on the building .
language action
b) (Optional Edit by Unlinking)
Semantics Diagram
ape jump on building
Canvas
building
house
Tap to locate original sketch in the canvas with a helpful arrow
c) (Optional Edit by Linking)
ape jump on building
Pen+touch sketch-to-word to remove sketch from command
d)
ape jump on building
Pen+touch a different sketch-to-word to link it to the command.
skyscraper building
e) (Confirmation — Tap again)
ape jump on building
language action
cloud cloud
fast
very
cloud
slow
ape
building house
skyscraper building
building
skyscraper building
The ape jumps onto the final building.

Figure 4: *Semantics Diagram*: An example workflow of error-correction (changing the target "building" from red house to yellow skyscraper). (a) spoken user input in the transcript panel. (b) The semantics diagram generated from the user's input to visualize system's interpretation. (c-d) Pen+touch interaction between diagram nouns and objects in the scene (including unlabeled ones) allows for re-linking to modify or correct the machine's selections. (e) Confirm language command. If a verb is unknown for a command, the user can pick from an auto-presented list of similar verbs, or otherwise cancel (Figure 27). In our implementation, if a sentence produces too long a diagram to fit in-view, the user can zoom-out independently from the canvas.



keep this part of the sentence remove this part
of the senten e keep this part of the sentence
remove this p of the sentence keep this part
of the sent

Figure 5: *Transcript View*: Selection / Deselection of Words: Toggle off/on words via touch-dragging to modify input for the next command. Small direct edits could be a desirable alternative to repeating the command verbally. For a fallback, the user can type with a keyboard to replace the text. (Left: quick operation buttons for selecting pieces of the text)

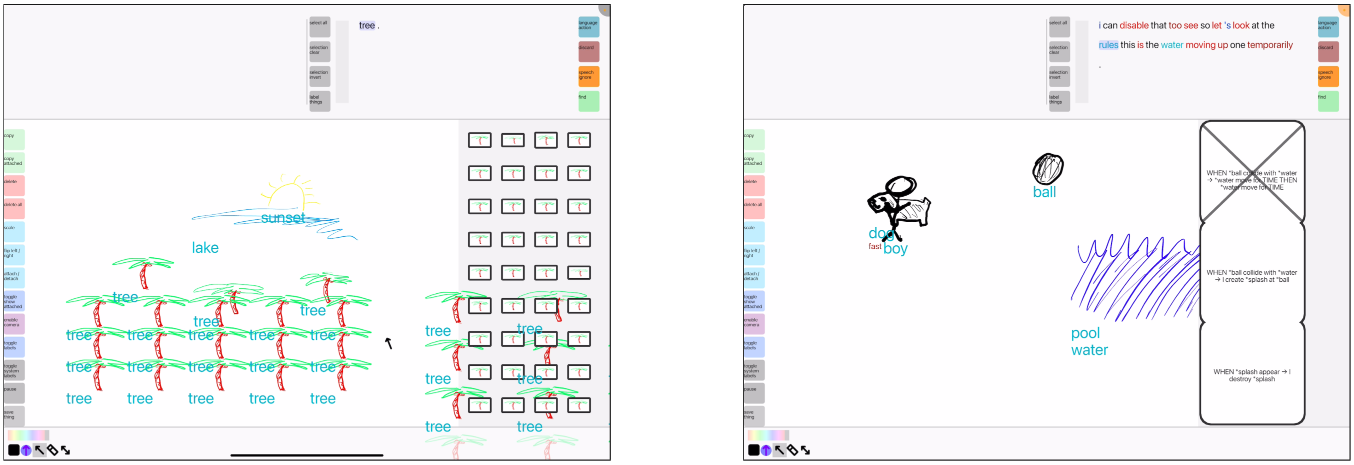
The following choose 1+ **specific** objects for a command by matching object labels (nouns, optional adjectives, etc.) with words:

- *1 or more objects: ... "the" <object label(s)> ...*
- *all objects: ... "all" <object label(s)> ...*
- *a number of objects: ... <number> <object label(s)> ...*
- *a single random object: ... "a"/"an" <object label(s)> ...*

There are a few special cases. The pronoun "I" is reserved; it allows the user to take-part in the narrative of a command (e.g. "I destroy the wall"). "Thing" is also a reserved noun that can refer to any object regardless of label. Plural nouns with no modifiers (e.g. as in "blades rotate") refer to labels used to define the interactions between objects with that label. Specifically, this is used to construct rules, as described below and in Figure 8.

Adjectives and adverbs are usable as labels that define the properties of objects. Verbs receive adjectives as arguments, which are evaluated continuously to modulate values and effects (e.g. magnitude, speed, size, distance, height). For example, *"fast", "slow," "excited"* impact jump height and/or movement speed. Adjectives also disambiguate like-noun labeled-objects (e.g. *"first house"* vs. *"second house"*). Adjectives can be removed programmatically via negation, e.g. *The thing is **not** fast.*

Adverbs heighten adjective effects multiplicatively – e.g. *"very"* and *"slightly"* – and are chainable – e.g. *"very, very."* Special adjectives offer system-control: e.g. "static" fixes sketches to the screen like a GUI element, useful for buttons, d-pads, score displays, etc.; "visible"/"invisible" toggle sketch visibility by command (e.g. *"The <...> becomes visible/invisible"*) or automatically when saying *"This is a(n) visible/invisible <thing...>"*.



tree .

i can disable that too see so let 's look at the rules this is the water moving up one temporarily

.

WHEN *ball collide with *water → *water move for TIME THEN *water move for TIME

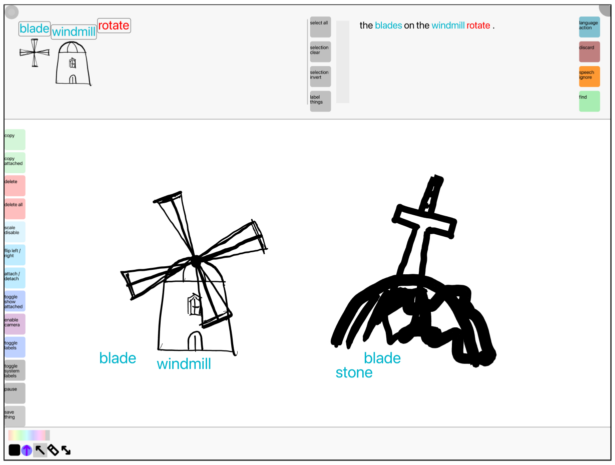
WHEN *ball collide with *water → I create *splash at *ball

WHEN *splash appear → I destroy *splash

(a) Finding Objects by Labels for Teleporting, Copying, Deleting

(b) Rule Toggling, Deleting

Figure 6: *Find Panel*: for performing search queries on sketches' noun and adjective labels. (a) Selecting the word "tree" searches for all objects with that label; tapping on an entry warps to the sketch; the eraser deletes the sketch instead; pen dragging copies entries (b) Forgetting active actions and rules: selecting the word "actions" (in the transcript) and an object in the scene will display a panel of all current actions affecting the object. Deleting an action using the pen will stop the action immediately. Selecting the word "rules" (in the transcript) will display active rules. Tapping with the pen will toggle the rule off/on.



the blades on the windmill rotate .

Figure 7: Hierarchical Disambiguation: Here, two sketches named "blade" are attached to different parent objects, but the system picks the one with the correct relationship by looking at the names in the objects' hierarchies. (The verb, "attach to" was used to attach the sketches beforehand.)

Rules are conditionals that run any kind of command in the future when objects with certain labels satisfy the condition. This enables the user to specify automated commands for the future "when," "as," or "after" an event has completed, without needing to know what they want or create objects in advance,

e.g., "*When* arrows collide with balloons, arrows destroy balloons.". Rules also allow definition of new verbs in terms of existing primitives, e.g. "flicker:" *When lights flicker, forever lights disappear for 0.1 seconds and then lights appear for 0.1 seconds.*

4.5 Examples

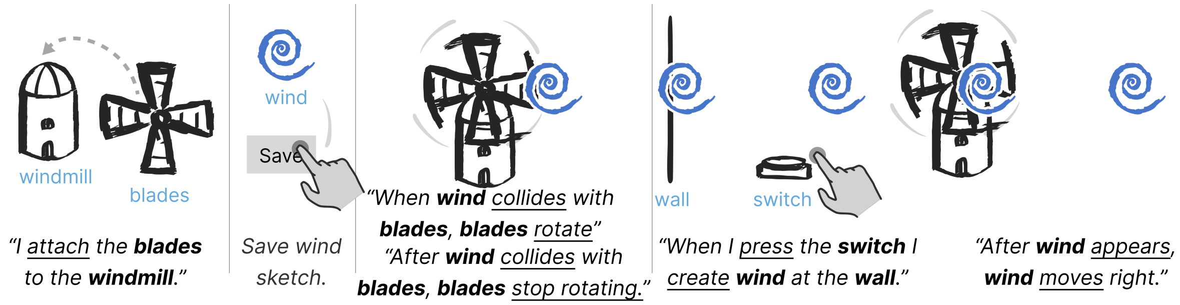
We demonstrate DrawTalking with illustrated examples showing multiple procedural and programming-like capabilities.

4.5.1 Pond scene. A simple example demonstrating randomized behavior is shown in Figure 1. Here a user sketches a frog, lily pads, water, and a butterfly. To cause the frog to hop randomly between any existing or future lily pad, they say: *"Forever the frog hops to* ***a*** *lily,"*. The butterfly is then commanded to follow the frog, and the user adjusts the speed of the butterfly: *"this butterfly is slow."*. The user can pause the simulation to edit any objects before resuming the actions.

4.5.2 Dog and boy's infinite game of fetch. This highlights sequences, loops, and user participation. The user sketches a boy, dog, water, and ball and commands the boy and dog to interact with the ball (Figure 3). The user can play the role of puppeteer by interactively moving the objects (e.g. the ball) to influence the other sketches' actions dynamically while the simulation is ongoing. For additional effect, the water could rise upon collision with the ball, with a command like: *"When water collides with balls water moves up for 0.2 seconds and then water moves down for 0.2 seconds."*

4.5.3 Windmill simulation. This example demonstrates rules, custom object saving and spawning, and buttons. (See Figure 8 for steps.) A key point is the flexibility to iterate on rules. In fact, here the rules can be defined in any order. If the user only defines the condition for on-collision, the blades will continue spinning by design. The user, however, can quickly rectify this by defining a rule to stop the blades *after* collision between wind and blades.

4.5.4 Creating the game "Pong" and turning it into "Breakout". This example shows how one can quickly reuse functionality to turn



"I attach the **blades** to the **windmill**."

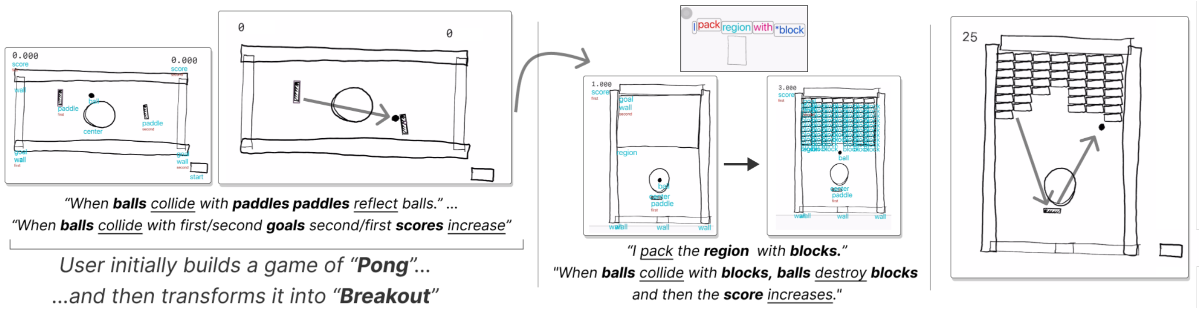
Save wind sketch.

"When **wind** collides with **blades**, **blades** rotate"
"After **wind** collides with **blades**, **blades** stop rotating."

"When I press the **switch** I create **wind** at the **wall**."

"After **wind** appears, **wind** moves right."

Figure 8: Windmill Simulation: Flexible process for constructing an interactive windmill built from user-defined rules, sketches, and triggers. The user can do this in *any* order and can try results at each step. This works on *any* sketch labeled "blade."

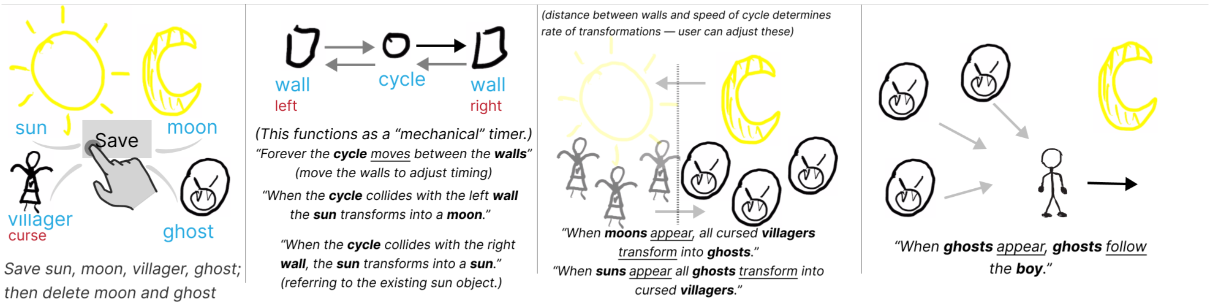


"When **balls** collide with **paddles paddles** reflect balls." ...
"When **balls** collide with first/second **goals** second/first **scores** increase"

User initially builds a game of "**Pong**"...
...and then transforms it into "**Breakout**"

"I pack the **region** with **blocks.**"
"When **balls** collide with **blocks**, **balls** destroy **blocks** and then the **score** increases."

Figure 9: Pong into Breakout: *Left:* per-player paddles and points; *Middle, Right:* scene reoriented, second player's objects removed, destructible bricks logic defined and objects added in-bulk into the desired region. (Note: not all commands shown.)



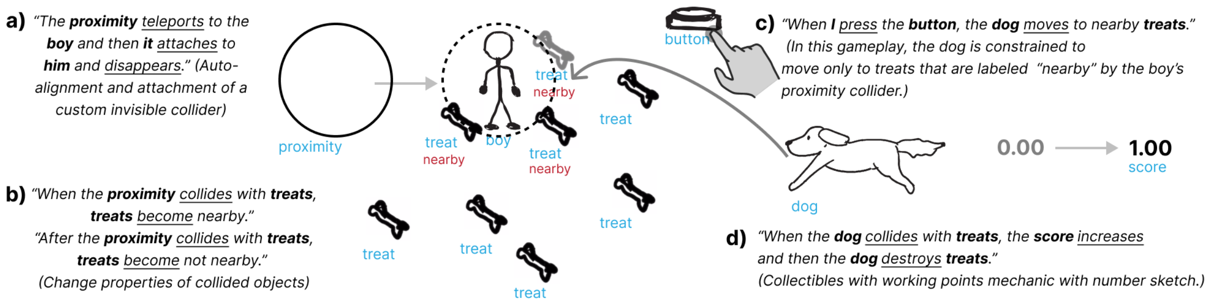
Save sun, moon, villager, ghost; then delete moon and ghost

(This functions as a "mechanical" timer.)
"Forever the **cycle** moves between the **walls**"
(move the walls to adjust timing)
"When the **cycle** collides with the left **wall** the **sun** transforms into a **moon**."
"When the **cycle** collides with the right **wall**, the **sun** transforms into a **sun**."
(referring to the existing sun object.)

(distance between walls and speed of cycle determines rate of transformations — user can adjust these)
"When **moons** appear, all cursed **villagers** transform into **ghosts**."
"When **suns** appear all **ghosts** transform into cursed **villagers**."

"When **ghosts** appear, **ghosts** follow the **boy**."

Figure 10: Day/Night Cycle with Transforming Villagers: A sun/moon transforms into the other in-time with collision events. When the moon appears, the villagers transform into ghosts, which chase the boy. This shows event-based behavior changes.



a) "The **proximity** teleports to the **boy** and then **it** attaches to **him** and disappears." (Auto-alignment and attachment of a custom invisible collider)

b) "When the **proximity** collides with **treats**, **treats** become nearby."
"After the **proximity** collides with **treats**, **treats** become not nearby."
(Change properties of collided objects)

c) "When **I** press the **button**, the **dog** moves to nearby **treats**."
(In this gameplay, the dog is constrained to move only to treats that are labeled "nearby" by the boy's proximity collider.)

d) "When the **dog** collides with **treats**, the **score** increases and then the **dog** destroys **treats**."
(Collectibles with working points mechanic with number sketch.)

Figure 11: "Dog Plays Fetch" as a Prototype Game Mechanic (demonstrating adjectives for dynamic logic): The dog interaction is re-imagined as an interactive game mechanic.

one idea into another, in this case, a version of the game "Pong" [1] into a version of the game "Breakout" [4] (Figure 9).

Create a ball and walls, then paddles, goals, and points for each player. For points, say *"I want the number 0"* and touch+pen the number to spawn a number object. Label with *"This is the first/second score, goal."*. Make points screen-space UI with *"This thing is static"* and set-up rules for collisions between the ball and goals: *"When balls collide with first/second goals second/first scores increase"*. Last, add collision logic: *"when balls collide with walls walls reflect balls"*, *"when balls collide with paddles paddles reflect balls."* Now we have a playable touch-based version of Pong.

We turn this into Breakout by rotating the canvas, deleting the second-player-related sketches, and adding a breakable blocks mechanic. To speed-up our process, we can sketch a temporary "region" and say *"I pack the region with blocks"* to fill it with sketches e.g. a custom "block" sketch. To make the blocks destructible and increase the point count, create a rule *"When balls collide with blocks balls destroy blocks and then the score increases."* We are done. In a few steps we have revised the scene into a different game.

4.5.5 Day/Night Cycle: Transforming Objects and Periodic Collision Logic. The example in Figure 10 demonstrates the ability to command objects to transform into other saved objects automatically, which not only changes objects' representation, but also enables automatic behavior and state changes. Further, the user can easily adjust periodic rates by using objects as a directly-manipulable, visual timing mechanism. By moving the walls in the figure closer or farther, we can opt to adjust the rate of transformation without the need for a precise timing value.

4.5.6 "Dog plays fetch revisited" as a playable gameplay mechanic. The dog interaction is re-imagined as an interactive game mechanic. The example demonstrates adjectives being used for logical control (Figure 11). The user hits a button to trigger the dog's fetch ability: collect a treat for points, but only if the treat is in the proximity of the boy's collider. This example demonstrates a mix of logical behavior-programming using adjectives, as well as the beginnings of quick game-prototyping iteration.

5 IMPLEMENTATION

5.1 Hardware and Software

We implemented DrawTalking natively on the iPad Pro M2, with the main code written in C++ and C, bridged with Objective C++ and Swift to access platform-specific APIs. Speech recognition is on-device. A local NodeJS+Python server-side for NLP runs on a MacBook Pro. Text is continuously sent to the server where 1) a dependency-tree is created, 2) sent to the client and compiled into a key-value format for semantic roles (e.g. *AGENT*, *OBJECT*), 3) either interpreted in real-time by our custom engine to drive interactive simulations, or used to find objects by deixis. In final stages, the system looks-up and maps objects to roles in the command structure and executes it to produce animations, effects, and state changes. (See Figure 12 for a diagram of the the concrete client-server implementation and data-flow.)

5.2 Processing Steps

The language processing component comprises roughly 4 steps that translate a natural language data structure into a command for the application (Figure 13). These user-initiated commands are scripts constructed and executed at run-time. As they're executed, the scripts handle data states, sequences, and dynamic instantiations of scripts corresponding to verbs in our built-in library. Verb scripts are modules comprising arbitrary code and accepted parameters, as opposed to generated code. The system draws from ideas in visual programming and virtual machine-based engines [3, 10, 31, 50]. Note that implementers can add functionality without modifying the rest of the system. For supplementary examples, see Appendix D.

Natural language raw text ⟶ S1. The system receives language input and outputs a directed graph data structure encoding the dependencies between words and the words' semantic roles.

Our implementation uses the Spacy (v 3.1.4)[45] library to output a dependency tree per-sentence, along with a library called "coreferee" to fill-in coreference information. Accuracy of this phase is tied to the chosen NLP method, not our interface. Spacy tended to reproduce the same results for the same sentence structures, which made visual output predictable.

S1 ⟶ S2. The system traverses S1 and generates a new generic graph structure S2. Each node entry in the graph structure represents a nested hierarchy of semantic units containing information such as labels (the word actually used) and part of speech (such as noun, verb, etc.).

Incomplete S2 ⟶ S2 with system-context feedback. Upon creating the structure for S2, there might be unspecified placeholders for objects, which we call "incomplete". The system will now look at application-context such as user-input and objects, and fill the structure with concrete entity IDs as-needed.

A query sub-system is needed to register and lookup objects based on their labels. For each noun-like entry in the traversal, the system queries for objects with the given noun and adjective labels, and returns the IDs of all objects in the world that match those labels.

Incomplete S2 ⟶ complete S2 with user feedback. After this process, the user should be able to do last-second editing and correction of the intermediate structure. The system can output user feedback (Figure 4).

S2 ⟶ S3. Lastly, the application traverses the structure and generates a final application-defined structure S3 that it can evaluate – in our case, a mix between scriptable virtual machine and animation engine akin to [48, 50] or a runnable node-based program [10]. S3 can call these scripts and retain S2 for reference. When traversing nodes in S2 labeled ACTION (the verbs) we lookup the appropriate previously-defined script for that action and insert a reference to it in S3. The arguments for that action, i.e. semantic role keys mapped to object ids or types, are inserted into a lookup table specific to the ACTION script instance so when the action executes, it knows what objects to modify. Loops and timed waits are also inserted into S3. The application evaluates the final structure according to its own interpretation. This results in any potential

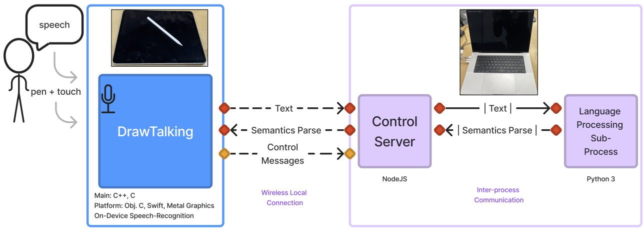
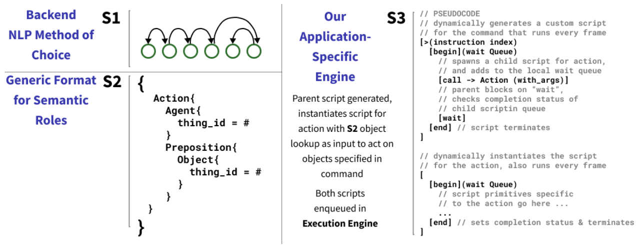

speech
pen + touch
DrawTalking
Main: C++, C
Platform: Obj. C, Swift, Metal Graphics
On-Device Speech-Recognition
Text
Semantics Parse
Control Messages
Wireless Local Connection
Control Server
NodeJS
| Text |
| Semantics Parse |
Inter-process Communication
Language Processing Sub-Process
Python 3

Figure 12: Concrete Client/Server Details: Text data are sent to a server for natural language processing over a data channel and forwarded to an NLP-focused sub-process on the same machine. The semantic parse data return to the client. A command channel handles user events independently (e.g. discarding commands interrupts in-flight stale NLP).


Backend NLP Method of Choice
S1
Generic Format for Semantic Roles
S2
Our Application-Specific Engine
S3

```
{
  Action{
    Agent{
      thing_id = #
    }
    Preposition{
      Object{
        thing_id = #
      }
    }
  }
}
```

Parent script generated, instantiates script for action with **S2** object lookup as input to act on objects specified in command

Both scripts enqueued in **Execution Engine**

```
// PSEUDOCODE
// dynamically generates a custom script
// for the command that runs every frame
[>(instruction index)
  [begin](wait Queue)
    // spawns a child script for action,
    // and adds to the local wait queue
    [call -> Action (with_args)]
    // parent blocks on "wait",
    // checks completion status of
    // child scriptin queue
    [wait]
  [end] // script terminates
]

// dynamically instantiates the script
// for the action, also runs every frame
[
  [begin](wait Queue)
    // script primitives specific
    // to the action go here ...
    ...
  [end] // sets completion status & terminates
]
```

Figure 13: Language processing into commands (S1 to S3): S1 is any implementation-specific NLP, e.g. a dependency tree; S2 stores generic semantic role labels created from S1; S3 is an app-specific engine that receives and interprets S2.

application-specific side-effect – in our case, animation, simulation, rule creation, application state-changes, and so on.

The application can also retain the generic S2 structure to generate commands later.

5.3 Current Limitations of Semantic Parsing

We focused on supporting narrative third-person form for this prototype due to our emphasis on scenarios involving explanation and storytelling. Passive tense, split infinitives, dangling prepositions, and idioms among others are unsupported. Other tenses and forms *are* supported as input, but behave the same as present tense. For example, past tense (*"the dog jumped"*), future tense (*"the dog will jump"*), present-progressive (*"the dog is jumping"*), and imperative (*"dog, jump"*) are equivalent to *"the dog jumps."* This allows for variety in the storytelling, but a future implementation might wish to distinguish between tenses for greater expressiveness.

Rules must sometimes be specified verbosely to avoid ambiguity in more complex cases. For example: *"When dogs collide with cats dogs jump and cats jump."* This could be expressed less verbosely, but the prototype at-present cannot disambiguate alternatives such as *"When dogs collide with cats they jump."* Here, "they" is ambiguous because it could refer to "dogs" only, "cats" only, or as intended in this example, both. The first form is more verbose, but unambiguous. We support the second, but just as in English, the result might be unclear and the system will likely pick either "dogs" or "cats."

Note that the semantics diagram still appears when a command is created from unsupported grammar. However, the diagram might show incorrect or missing object mappings and semantic roles. In such cases, the user can discard the malformed command or link missing objects to the diagram as in Figure 4. The user must learn and restrict themselves to the specific grammar to get the most consistent results. We cover how we might support more natural speech in section 8.

Although commands are user-initiated by design, a potential limitation is that commands must be created and confirmed one-by-one. Continuous speech input has unclear sentence boundaries, so we do not currently integrate a component to predict where a command should end because this could be unreliable and would take away control. To avoid rushing the user, we did not rely on a timeout either. We chose to give the user control over the sentence boundaries via the speech command button and interaction with the text transcript view (Figure 5). Different implementations could try automating commands without compromising user-control.

Additionally, the procedure to transform S1 into S2 (section 5.2) is coupled to the NLP library of choice by design. S2 and onwards are generic. An implementer would need to be aware of the need to build their own version of this translation step if their NLP back-ends use different dependency labels.

6 OPEN-ENDED USER STUDY

We conducted an exploratory discussion-focused study to gauge understandability of interaction, discover use cases and directions, and learn users' perceptions. We believe this form of verbal feedback is valuable because it can extend our own understanding of the tool with specific anecdotal and experiential feedback. The goal was to give the participant enough exposure to the interface to arrive at working artifacts from the process and elicit meaningful discussion.

To that end, the study was an open-ended exploration between the researcher and the participant. To mitigate the potential for biasing participants towards certain answers, the researcher did **not** reveal the design goals or indicate particular use cases for the tool. Participants were informed only that the tool was a prototype with animation features controlled by sketching and speaking. We let participants draw their own conclusions as to the usefulness, use cases, and potential for the interactions.

We chose this approach because we were interested in qualitative, early stage feedback on the *concept* and *interactions* behind DrawTalking, rather than feedback on our implementation, user performance, or based on comparisons. There is no known baseline for comparison with DrawTalking's interactions, and at this stage, we felt it would not contribute towards achieving the goals of our research to compare the DrawTalking prototype with production tools in terms of usability (i.e. the "usability trap" as in Olsen's work [46]). We wanted user feedback on potential use cases based on discussions rather than specializing on a specific use case.

6.1 Participants and Procedure

We invited 9 participants (students, professors, artists, game design) chosen by responses to an online recruitment form. We wanted candidates with relatively-high confidence in speaking and sketching, with interest in the topic, and with a fair mix of professional/academic backgrounds. For participant details, see Appendix B.

Each study lasted 1 hour and 15 minutes, and each participant was compensated by 30 USD. For each session, the researcher sat alongside the participant in front of DrawTalking running on an iPad, and taught the drawing features. Next, the participant was told to draw 5 objects of their choice, then taught both object labeling methods. Then, the session was improvised using the objects to explore all language features in rough increasing complexity. The researcher could help or suggest ideas, but participants mainly guided their own exploration using a provided language features list (the Info Sheet supplemental material), and each experience was different and ended with different interactive scenes. Participants were allowed to think-aloud and comment. After, the participant was asked to reflect on the experience. The researcher ended by showing additional pre-recorded videos and soliciting final feedback. We analyzed the results by taking note of common key words (e.g. fluidity) and of unique anecdotes, suggestions, and comments from the session and post-discussion. We recorded each session and took screen-shots for post-viewing (For samples, see subsection B.2). In the following section, we report the discussion-oriented results.

7 RESULTS AND DISCUSSION

All participants understood and learned the mechanics, and produced thoughtful discussion. They tried their own ideas for how to make things or test their own understanding out of curiosity. *All became comfortable with the core mechanics (controls, touch+deixis, commands, etc.) in 10-15 minutes. Some were excited to continue playing with the tool.* They identified use cases including educational applications, rapid video game and design prototyping, paper prototyping, user interface design, presentation, language learning, and visual-oriented programming.

We explore the qualitative experiences of participants using DrawTalking. The following come from the open-ended discussions and roughly covers topics related to the design goals.

7.1 As programming-like capability

DrawTalking's style of control shows promise for empowering both programming and non-programming audiences (D4).

Several participants with programming knowledge and experience teaching programming compared DrawTalking functionality to constructs like variables, loops, and conditionals, indicating the patterns were familiar (D4). Some were reminded of Scratch [34] if it did not have the explicit programming interface.

In contrast, P7 (an experienced digital artist without programming experience) appreciated DrawTalking as an accessible tool for non-programmers that could reduce frustration encountered with existing programming tools: *"Me instructing a game engine – someone like me who's not a programmer and who is intimidated by doing C# or ... using [a] visual scripting language like blueprints – this is a really clean interface that I think can achieve you can get 90- or 80% the way there. It just makes the user experience cleaner than having to use all these knobs and buttons or things like that or using scripting language or having to actually write code. You [the researcher] not force me to work with the scripting language. "*. P7 here favors the *simplicity* and *accessibility* of language control as a complement to more powerful programming interfaces.

The takeaway of interest is that **not including an explicit coding interface was appealing for its simplicity and accessibility.** Nevertheless, it would be interesting to explore balancing a greater level of complexity (in terms of interface) to serve more expert users, without compromising the simplicity for non-programmers. P7, for example, is aware of more powerful tools, but prefers something simpler. DrawTalking, in concept, seems to serve the role as abstraction layer between language- and code-based control. We could think about different modes of information-hiding depending on the task and audience.

7.2 Semantics-visuals correspondence is valued

Participants valued the visual correspondence between visual elements and language embedded in DrawTalking's sketches and components as a strength in the design. (P5: *"Mixing visual elements with language helps me understand it. More of that [in future explorations] could be good... drawing connections between [things] visually as I'm speaking."*)

For the semantics diagram, participants understood it as a helpful means to introspect and correct the system's understanding of user input. P2: *"[The diagram] is pretty important. Especially if you're getting to used to the program, you might say something wrong and it interprets it in a different way and you can correct it."* P8: *"This is a really cool debug thing. I immediately understand what happens when I read it."* Additionally, the interface *simplifies* the input, making it easier to read: P7: *"[It's] showing what it's going to do. It's taking the verbs and cutting all the other stuff out."* The feedback suggests that **the diagram is valuable for instilling confidence in the system**, and that the interface achieved D3.

For the transcript view, we observed users (e.g. P1, P5) habitually clearing the transcript using "discard" on their own. We asked why and some participants reported wanting to keep the interface clean. We did not foresee this, but this user behavior serves a dual-purpose: the user is aware of what they say, which means they are perhaps likelier to catch errors, and they also help by keeping the system input from growing too long. We can take this as a positive: the system helps the user, and the user helps the system. We can also take this as room for improvement: how to keep the transcript view easier to clean without user intervention?

7.3 Physicality, Prototyping, Playfulness

A reported strength of DrawTalking was the combination of physicality with logic and animation: P5: *"I can execute different rules of things that are happening. The scene is playing out here AND physical thing is happening."* P2, a games student, likened DrawTalking to a form of digital, semi-automatic *paper-prototyping*: *"When we make games there's something called a paper prototype where we make a bunch of pieces of the players and the objects. We just move it around by hand to kind of simulate what it would be. [DrawTalking] is kind of like that but on steroids a bit. So it's very nice to be able to kind of have those tools to help you with it without needing to manually make it."* Relatedly, P1 said, *"It's like language and Legos put-together."*

This suggests that **the design decision to give users interactive control via spatial direct manipulation in tandem with**

automation was successful; it enabled a form of playful exploration. This supports something akin to what professional designers and kids use to tinker with creative ideas. Note that P2 values "help with making," but doesn't ask for full automation. To them, interactivity was important for the playful prototyping feel.

7.4 Creative problem-solving

P1 was able to discover creative use of rule functionality by their own curiosity. In their case, they drew a boy, house, and tree. By creating two co-dependent rules, they wanted to see if they could recreate infinite loop functionality without using "over and over" or "forever." Their solution was *"When boys collide with trees, boys move to houses"* and *"When boys collide with houses, boys move to trees."*. This resulted in an infinite loop once the boy sketch collided with either the tree or the house. The example demonstrates imaginative composition of language-controllable primitives (Figure 16).

A highlight in P2's session involved working-around a then-limitation of one of the verbs, "climb." The scene contained a tree and squirrel, and we wanted the squirrel only to climb the tree if it were not already on-top. However, our simple version of "climb" was (at the time) programmed simply to move an object to the base of a target and then move it to the top, so the squirrel would inadvertently climb down then up. To work around this issue, a solution was devised in which an invisible collision box placed at the top of the tree programmed with *"When squirrels collide with the collider, squirrels stop climbing."*. Then, upon *"The squirrel climbs the tree"*, the discovery was made that the collider could also be used as an in-situ stopping mechanism the user could simply drag onto the squirrel at any time to stop it (Figure 17).

We believe this means DrawTalking's interactions **successfully enable creativity, discoverability, and playful problem-solving, independent of the limitations of the featureset.**

7.5 Re-applicability to other toolsets

Users saw potential for DrawTalking as generic functionality for integration with other applications in a creative pipeline, e.g. as a playful ideation phase from which to import/export assets or runnable code; attaching speech control to production tools to support faster workflows. P1: *"You could sit here and have a conversation and build up just using language your interactions and then you send that out for [exporting] code."* P3: (comparing with an Adobe After-Effects workflow) *"here, it just takes one second for me to 'say it' – [this could be a] built-in function for any tool/interactive AI"*. P3 felt the interactions were fluid. P8 emphasized language control: *"just incorporate the language and control interface here. We can easily create animations with the fancy stuff they have."*

In other words, **participants perceive DrawTalking as an interaction that is independent from a specific application.** The suggestions for outputting code or plugging-into other applications with more advanced functionality hint at an interest in DrawTalking as a general control mechanism. Participants expressed excitement over how the ideas might be re-applied.

7.6 Observations and Feedback on Current Limitations of Language Understanding

Participants occasionally encountered some of the limitations of the semantic parsing described in subsection 5.3. For instance, P2 initially said, *"When the squirrel collides with the collider stop."* The prototype cannot currently infer that the rule should stop the squirrel. The subject (squirrel) needs to be re-specified as the target object to stop. The correct form would be: *"When the squirrel collides with the collider the squirrel stops."*. Similarly, P1 tried, *"The boy moves to the house and then to the tree."* The prototype does not infer the second subject and verb, so the input must be: *"The boy moves to the house and then the boy moves to the tree."* Other user errors included using imperative form without a subject (P5 *"Make a star"*) (as imperatives currently require an explicit object), or expecting alternative behavior or synonyms (e.g. "touch" as a substitute for "collide with" or a variant of "follow" that moves to a target only once, not continuously). Most of the unsupported cases suggest the following categories for improvement in future prototypes: reducing the need for user-specification, even in ambiguous or less-grammatically-correct cases; inferring implicit meaning from context; making the vocabulary and tenses more flexible. These are not tied to the DrawTalking concept, but could improve the experience of a concrete implementation.

As for the study experience, participants needed to become acquainted with the supported language structures. This was expected in the natural process of learning a new interface. In spite of the limitations, participants did not noticeably express frustration; they were excited. We suspect this is due to the following main factors:

Interaction design: invalid commands do not cause surprising or destructive side-effects (nothing happens) and redoing commands is fast. Pressing the discard button and speaking again does not incur too much of a time cost before the participant can see results. The world isn't interrupted; it continues running and is interactive even while staging a command. In short, the quick and fluid error recovery might have helped reduce wait-time.

Participants were able to focus on the potential of the interactions and design: they were informed of the fact that they were being introduced to a prototype and they formulated responses with this in-mind. This is evidenced in the fact that each participant provided insight on potential improvements, integrations, and use cases assuming a more robust system. P1 and P5 especially, as seasoned computer scientists and educators, focused on the potential to integrate the workflow, feel, and functionality into other software.

This increases our confidence that participants focused their feedback on the concept rather than the implementation details.

7.7 Naming by speech preferred to text linking

Labeling by speech (deixis) was unanimously preferred to linking and was used most often or exclusively. (See subsection 4.3 to recap labeling functionality.) It was considered more intuitive, direct, and similar to how people talk. On the other hand, pen+touch linking could be useful when freer speech (without explicitly naming objects) is preferred, e.g. for flexibility while giving talks or to reference previous discussion without repeating words. Labeling by linking directly to a word instead of by speaking could allow for freer use of language, as the desired labels just need to be present in the narration. P5 (an interactive computer graphics professor) gave such as example: *"I often talk about arrows in the context of vectors in my teaching"*. Here, no deixis is used. Instead, the user would perform a pen+touch operation between objects and the words "arrows" and "vectors" to do the labeling. The speaker can perform this labeling in the background without needing to use a specific grammar.

However, if the user does prefer the speech approach to the linking approach, this means it might be possible to reduce reliance on the transcript view, which could open-up opportunities to hide the transcript for contexts when it might be inconvenient. e.g. for eyes-free interfaces or XR spatial systems.

7.8 User control of physical elements and robots

Participant P9 was a roboticist who could give specialized feedback. When asked about possibly using DrawTalking for controlling robots, P9 said that for real-world environments, they might want to have even **more control and self-specification of commands** due to safety concerns, in spite of the loss of automation:

P9: For potential tangible *interfaces, physical things like robots might need more specification (for safety? accuracy?). [I] would be more tolerant of extra specification to make sure it's correct. Definitely don't want surprises. People won't want it to be fully magical. [They want] more control.*

This suggests that our emphasis on user control is likely important for reapplication of DrawTalking concepts in use cases involving physical objects. Additional safety considerations emerge and there are trade-offs between automation and control.

8 FUTURE WORK

In sum, users described DrawTalking as fluid and flexible; a natural language way of achieving programming-like functionality; a rapid prototyping environment; an independent general interaction technique; capable of integration with other applications; physical, tangible, spatial; accessible to kids; a new approach to working-out visual problems.

We believe DrawTalking works, then, because it successfully captures some of the playful and creative attributes of programming, spatial manipulation and sketching and language, owing to our initial goals and to our designs.

Based on the primary contribution of our work (the interaction mechanism enabling sketching+speaking control) concrete implementations could use the concept to drive deeper levels of functionality and programmability than what is capable with our application prototype. The overarching idea, above all, is about exploring how to provide controllable feedback and facilitate a kind of *vernacular* style of programming, to extend our creative capabilities.

There are many directions: improving the system design and vocabulary; further-exploring visual-linguistic mappings; integrating with other applications; longer-term studies applying our approach to domain-specific creative or educational workflows; multi-user collaborations; application to spatial experiences for interaction with the real-world. We describe several future possibilities:

Integration with other applications and extensibility. A DrawTalking-like sketching application could be a part of a prototyping/open-ended phase within a creative process. Future work can explore how to take the interaction with named rough sketches and convert them for use or replacement in later-stage production phases, in which goals have become more specific. The names and even the history of users' direct manipulation and narration could be used to inform such a conversion process by providing context. We can also explore how to convert back and forth between different representations of objects and behaviors to support moving seamlessly between ideation and production phases. For example, we suspect that our strategy of naming objects independent of their representation could be useful as a way of generating scenes from the sketch representation.

Additionally, as P9 (section 6) suggests, DrawTalking could either provide animations to other tools, or else control external tools' capabilities (not just limited animations) via plugin or remote API. This way, we treat DrawTalking as a controller rather than as a standalone application. (This also helps avoid vendor lock-in.) Vocabulary and functionality are potentially unlimited.

Deeper Exploration of relationships and mappings between semantics and visuals. There are several unexplored concepts relevant to semantics-visual mapping. For example, the use of sketches to represent abstract concepts could be evaluated as additional means for controlling content, in addition to the concrete sketches and language we've used. The semantics diagram, further, could be extended to support greater levels of editability. Another future area might involve context-sensitivity: What if the user's culture and assumptions should produce different behaviors and visual-semantics mappings? How might DrawTalking evolve when considering not only English, but also other languages, modes of communication, and cultural contexts?

Supporting language input that is more natural. Future work could work towards achieving increasingly natural and free-flowing speech input for interactive interfaces by implementing a more robust and feature-complete language processing and multi-modal input back-ends. We suspect that a translation layer from natural, more fragmented speech could convert input into simpler, structured, and deterministic instructions similar to what our interface uses. This direction could allow existing interfaces with deterministic instructions like ours to accept more expressive, less restricted language input, independent of their functionality. This could lead to reduced user-specification and a better approximation fully "natural" language interfaces.

Making trade-offs: reasons why future interface designers might not want to use or rely on generative-artificial-intelligence models (e.g. LLMs), and why we did not use them. To achieve fully natural input, a suggestion might be to use generative AI models such as large language models (LLMs) to perform the aforementioned proposed translation of language input. These models could also generate visual content alongside user-drawn sketches. (A number of study participants made this suggestion.)

However, as per our design goals for this interface (subsection 3.3), the system *must* be interactive and instant, and the user must be in-control of the content's representation. We considered LLMs, but they contradicted our design goals: reducing fluidity, user control over content, and system transparency and understandability. Interface designs with similar criteria might also encounter the same challenges and opt to make trade-offs.

LLMs are currently too slow to provide instant feedback, even if we use LLMs only to transform natural input into structured commands as we suggest. (This also raises additional research questions: How well can LLMs rewrite users' speech and reflect intent?) In contrast, the NLP technology we chose responds nearly instantly.

Also, LLMs often exhibit unpredictable or unreproducible behavior, whereas the chosen NLP returns the same results for the same language input. We wanted determinism so command outcomes would be easy to reason about, especially to keep the scope of possibility manageable for the user study. Not using LLMs allowed for feasibility and stable development.

Still, generative-AI is promising given the right context and requirements. Future work could see how LLMs might be complementary to other methods (traditional NLP, procedural generation, and others). We believe a good direction for research would be to find the balance between the many approaches and solutions we have developed over time.

Spontaneous authoring of new primitives without programming. Exploring interactions for richer behavior-authoring interactions compatible with our approach would be another direction. We can imagine a number of possibilities: exploring extensions to the existing semantics diagram interface for lightweight editing; using DrawTalking's controls to drive other applications with their own unique primitives; combining procedural methods with the capabilities of generative models by finding the right balance; crowdsourcing programmed-functionality through shared libraries of domain-specific functionality; introducing multiple layers of programming capability in the same interface for different audiences (e.g. artists, programmers), trading-off simplicity for granular control as is common in many applications.

Possibilities for generalizability and scalability. We envision that combinations of DrawTalking-like functionality and different implementations and domain-specific applications could lead to a more-scalable ecosystem of interfaces. We also believe that by decoupling natural language processing implementations from the interface (rather than building directly around specific NLP implementations or probabilistic results) it might be easier to support reproducible research within interactive interface design. This is in-part why we chose to compile NLP information into intermediary, generic command structures. These could feasibly be used in future research as a compilation target by any other NLP or alternative approach without relying on a specific implementation. For example, this is where a gen-AI model could be explored without changing the rest of the system.

Lastly, although we've focused on rough sketching, conceptually there is no limitation preventing sketches from being replaced with other forms of controllable digital or tangible objects. Sketches can represent collections of variables and properties mapped to some objects, whatever forms they might take.

We are excited to see ideas taken in new directions for playful and creative human-machine interfaces beyond our initial scope. We think that there is an opportunity to seek-out new human-computer interactions that draw from our natural behavior for inspiration.

9 CONCLUSION

We have introduced DrawTalking, an approach to building and controlling interactive worlds by sketching and speaking while telling stories, and have shown its potential. Our interface was inspired by our natural interplay between sketching and language, and our ability to communicate via make-believe. There are many possible directions, and we are excited to see future research build on our approach and uncover other human-centered approaches to extending natural human abilities. We consider this project just one possible step and we hope that it will foster fruitful discussion and research.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers (of this version and an earlier CHI LBW version of this work [51]) for their constructive comments; Professor Lakshminarayanan Subramanian for his feedback; Devamardeep Hayatpur for additional feedback on writing and visuals for the CHI LBW; Daniel Zhang for brainstorming and discussions; the user study participants for their time and insights; and the many others who participated in conversations and gave insights during the journey of this work.

REFERENCES

[1] 2022. 50 Years of Fun With Pong. https://computerhistory.org/blog/50-years-of-fun-with-pong/ Section: Curatorial Insights.

[2] Maneesh Agrawala, Wilmot Li, and Floraine Berthouzoz. 2011. Design principles for visual communication. *Commun. ACM* 54, 4 (April 2011), 60–69. https://doi.org/10.1145/1924421.1924439

[3] Another World [n. d.]. Another World Code Review. https://fabiensanglard.net/anotherWorld_code_review/

[4] AtariAdmin. 2022. New Insight into Breakout's Origins. https://recharged.atari.com/the-origins-of-breakout/

[5] bill wurtz. 2016. history of japan. https://www.youtube.com/watch?v=Mh5LY4Mz15o

[6] bill wurtz. 2017. history of the entire world, i guess. https://www.youtube.com/watch?v=xuCn8ux2gbs

[7] Richard A. Bolt. 1980. Put-that-there: Voice and gesture at the graphics interface. *ACM SIGGRAPH Computer Graphics* 14, 3 (July 1980), 262–270. https://doi.org/10.1145/965105.807503

[8] K. Compton and M. Mateas. 2015. Casual Creators. https://www.semanticscholar.org/paper/Casual-Creators-Compton-Mateas/f9add8f5126faab72c9cc591b5fdc7e712936b56

[9] Bob Coyne and Richard Sproat. 2001. WordsEye: an automatic text-to-scene conversion system. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques (SIGGRAPH '01)*. Association for Computing Machinery, New York, NY, USA, 487–496. https://doi.org/10.1145/383259.383316

[10] cycling74. 2018. *Max*. https://cycling74/products/max

[11] Richard C. Davis, Brien Colwell, and James A. Landay. 2008. K-sketch: a 'kinetic' sketch pad for novice animators. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. Association for Computing Machinery, New York, NY, USA, 413–422. https://doi.org/10.1145/1357054.1357122

[12] Griffin Dietz, Jimmy K Le, Nadin Tamer, Jenny Han, Hyowon Gweon, Elizabeth L Murnane, and James A. Landay. 2021. StoryCoder: Teaching Computational Thinking Concepts Through Storytelling in a Voice-Guided App for Children. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–15. https://doi.org/10.1145/3411764.3445039

[13] Dreams 2020. Dreams. [Playstation 4]. https://www.mediamolecule.com/games/dreams https://indreams.me.

[14] Judith E. Fan, Wilma A. Bainbridge, Rebecca Chamberlain, and Jeffrey D. Wammes. 2023. Drawing as a versatile cognitive tool. *Nature Reviews Psychology* 2, 9 (Sept. 2023), 556–568. https://doi.org/10.1038/s44159-023-00212-w Number: 9 Publisher: Nature Publishing Group.

[15] Adele Goldberg and David Robson. 1983. *Smalltalk-80: the language and its implementation.* Addison-Wesley Longman Publishing Co., Inc., USA.

[16] Forrest Huang, Eldon Schoop, David Ha, and John Canny. 2020. Scones: towards conversational authoring of sketches. In *Proceedings of the 25th International*

Conference on Intelligent User Interfaces (IUI '20). Association for Computing Machinery, New York, NY, USA, 313–323. https://doi.org/10.1145/3377325.3377485

[17] Jennifer Jacobs, Joel R. Brandt, Radomír Meěh, and Mitchel Resnick. 2018. Dynamic Brushes: Extending Manual Drawing Practices with Artist-Centric Programming Tools. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems (CHI EA '18)*. Association for Computing Machinery, New York, NY, USA, 1–4. https://doi.org/10.1145/3170427.3186492

[18] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: sketching dynamic and interactive illustrations. In *Proceedings of the 27th annual ACM symposium on User interface software and technology (UIST '14)*. Association for Computing Machinery, New York, NY, USA, 395–405. https://doi.org/10.1145/2642918.2647375

[19] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014. Draco: bringing life to illustrations with kinetic textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Association for Computing Machinery, New York, NY, USA, 351–360. https://doi.org/10.1145/2556288.2556987

[20] Khan Academy. 2014. DNA | Biomolecules | MCAT | Khan Academy. https://www.youtube.com/watch?v=AmOO4j0E408

[21] Khan Academy. 2017. Organelles in eukaryotic cells | The cellular basis of life | High school biology | Khan Academy. https://www.youtube.com/watch?v=bWPQvxElpLY

[22] Yea-Seul Kim, Mira Dontcheva, Eytan Adar, and Jessica Hullman. 2019. Vocal Shortcuts for Creative Experts. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300562

[23] James A. Landay. 1996. SILK: sketching interfaces like krazy. In *Conference Companion on Human Factors in Computing Systems (CHI '96)*. Association for Computing Machinery, New York, NY, USA, 398–399. https://doi.org/10.1145/257089.257396

[24] Gierad P. Laput, Mira Dontcheva, Gregg Wilensky, Walter Chang, Aseem Agarwala, Jason Linder, and Eytan Adar. 2013. PixelTone: a multimodal interface for image editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. Association for Computing Machinery, New York, NY, USA, 2185–2194. https://doi.org/10.1145/2470654.2481301

[25] Joseph J. LaViola and Robert C. Zeleznik. 2004. MathPad2: a system for the creation and exploration of mathematical sketches. In *ACM SIGGRAPH 2004 Papers (SIGGRAPH '04)*. Association for Computing Machinery, New York, NY, USA, 432–440. https://doi.org/10.1145/1186562.1015741

[26] Bongshin Lee, Rubaiat Habib Kazi, and Greg Smith. 2013. SketchStory: Telling More Engaging Stories with Data through Freeform Sketching. *IEEE Transactions on Visualization and Computer Graphics* 19, 12 (Dec. 2013), 2416–2425. https://doi.org/10.1109/TVCG.2013.191 Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[27] Germán Leiva, Jens Emil Grønbæk, Clemens Nylandsted Klokmose, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2021. Rapido: Prototyping Interactive AR Experiences through Programming by Demonstration. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*. Association for Computing Machinery, New York, NY, USA, 626–637. https://doi.org/10.1145/3472749.3474774

[28] Toby Jia-Jun Li, Marissa Radensky, Justin Jia, Kirielle Singarajah, Tom M. Mitchell, and Brad A. Myers. 2019. PUMICE: A Multi-Modal Agent that Learns Concepts and Conditionals from Natural Language and Demonstrations. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 577–589. https://doi.org/10.1145/3332165.3347899

[29] Jian Liao, Adnan Karim, Shivesh Singh Jadon, Rubaiat Habib Kazi, and Ryo Suzuki. 2022. RealityTalk: Real-Time Speech-Driven Augmented Presentation for AR Live Storytelling. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3526113.3545702

[30] Little Big Planet 2008. Little Big Planet. [Playstation 3]. https://www.mediamolecule.com/games/littlebigplanet

[31] Little Big Planet 2 2011. Little Big Planet 2. [Playstation 3]. https://www.mediamolecule.com/games/littlebigplanet2

[32] Xingyu Bruce Liu, Vladimir Kirilyuk, Xiuxiu Yuan, Peggy Chi, Xiang 'Anthony' Chen, Alex Olwal, and Ruofei Du. 2023. Visual Captions: Augmenting Verbal Communication with On-the-fly Visuals. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI)*.

[33] Edward Loper and Steven Bird. 2002. NLTK: the Natural Language Toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1 (ETMTNLP '02)*. Association for Computational Linguistics, USA, 63–70. https://doi.org/10.3115/1118108.1118117

[34] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. *ACM Transactions on Computing Education* 10, 4 (Nov. 2010), 16:1–16:15. https://doi.org/10.1145/1868358.1868363

[35] Michael New. 2015. The Circle of Fifths - How to Actually Use It. https://www.youtube.com/watch?v=d1aJ6HixSe0

[36] George A. Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (Nov. 1995), 39–41. https://doi.org/10.1145/219717.219748

[37] MinuteEarth. 2021. Why These Bears "Waste" Food. https://www.youtube.com/watch?v=b0dabXAy7uA

[38] minutephysics. 2013. Immovable Object vs. Unstoppable Force - Which Wins? https://www.youtube.com/watch?v=9eKc5kgPVrA

[39] minutephysics. 2014. Antimatter Explained. https://www.youtube.com/watch?v=Lo8NmoDL9T8

[40] minutephysics. 2021. The Physics of Windmill Design. https://www.youtube.com/watch?v=WGKIjojADmg

[41] MIT OpenCourseWare. 2008. Lec 1 | MIT 5.111 Principles of Chemical Science, Fall 2005. https://www.youtube.com/watch?v=2x3F08_8B80

[42] MIT OpenCourseWare. 2019. 1. Radiation History to the Present — Understanding the Discovery of the Neutron. https://www.youtube.com/watch?v=7LyvAVjQUR8

[43] MIT OpenCourseWare. 2019. 11. Radioactivity and Series Radioactive Decays. https://www.youtube.com/watch?v=z_xyx-z6arc

[44] MIT OpenCourseWare. 2019. Ep. 6: Element Production (Fusion) – Part 1. https://www.youtube.com/watch?v=zqXBZ81bWOc

[45] Ines Montani, Matthew Honnibal, Matthew Honnibal, Sofie Van Landeghem, Adriane Boyd, Henning Peters, Paul O'Leary McCann, jim geovedi, Jim O'Regan, Maxim Samsonov, György Orosz, Daniël de Kok, Duygu Altinok, Søren Lind Kristiansen, Madeesh Kannan, Raphaël Bournhonesque, Lj Miranda, Peter Baumgartner, Edward, Explosion Bot, Richard Hudson, Raphael Mitsch, Roman, Leander Fiedler, Ryn Daniels, Wannaphong Phatthiyaphaibun, Grégory Howard, Yohei Tamura, and Sam Bozek. 2023. explosion/spaCy: v3.5.0: New CLI commands, language updates, bug fixes and much more. https://doi.org/10.5281/zenodo.7553910

[46] Dan R. Olsen. 2007. Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology (UIST '07)*. Association for Computing Machinery, New York, NY, USA, 251–258. https://doi.org/10.1145/1294211.1294256

[47] Sharon Oviatt. 1999. Ten myths of multimodal interaction. *Commun. ACM* 42, 11 (Nov. 1999), 74–81. https://doi.org/10.1145/319382.319398

[48] Ken Perlin and Athomas Goldberg. 1996. Improv: a system for scripting interactive actors in virtual worlds. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques (SIGGRAPH '96)*. Association for Computing Machinery, New York, NY, USA, 205–216. https://doi.org/10.1145/237170.237258

[49] Ken Perlin, Zhenyi He, and Karl Rosenberg. 2018. Chalktalk : A Visualization and Communication Language – As a Tool in the Domain of Computer Science Education. https://doi.org/10.48550/arXiv.1809.07166 arXiv:1809.07166 [cs].

[50] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: programming for all. *Commun. ACM* 52, 11 (Nov. 2009), 60–67. https://doi.org/10.1145/1592761.1592779

[51] Karl Toby Rosenberg, Rubaiat Habib Kazi, Li-Yi Wei, Haijun Xia, and Ken Perlin. 2024. DrawTalking: Towards Building Interactive Worlds by Sketching and Speaking. In *Extended Abstracts of the 2024 CHI Conference on Human Factors in Computing Systems (CHI EA '24)*. Association for Computing Machinery, New York, NY, USA, Article 113, 8 pages. https://doi.org/10.1145/3613905.3651089

[52] J. Ross, Oliver Holmes, and Bill Tomlinson. 2012. Playing with Genre: User-Generated Game Design in LittleBigPlanet 2. https://www.semanticscholar.org/paper/Playing-with-Genre%3A-User-Generated-Game-Design-in-2-Ross-Holmes/75f36eb8585d9d7039a98c750b0085cc973eb689

[53] Nazmus Saquib, Rubaiat Habib Kazi, Li-yi Wei, Gloria Mark, and Deb Roy. 2021. Constructing Embodied Algebra by Sketching. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3411764.3445460

[54] John Sarracino, Odaris Barrios-Arciga, Jasmine Zhu, Noah Marcus, Sorin Lerner, and Ben Wiedermann. 2017. User-Guided Synthesis of Interactive Diagrams. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 195–207. https://doi.org/10.1145/3025453.3025467

[55] Jeremy Scott and Randall Davis. 2013. Physink: sketching physical behavior. In *Adjunct Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 9–10. https://doi.org/10.1145/2508468.2514930

[56] Yang Shi, Zhaorui Li, Lingfei Xu, and Nan Cao. 2021. Understanding the Design Space for Animated Narratives Applied to Illustrations. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems (CHI EA '21)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3411763.3451840

[57] Andreea Stapleton. 2017. Deixis in Modern Linguistics. *Essex Student Journal* 9, 1 (Jan. 2017). https://doi.org/10.5526/esj23 Number: 1 Publisher: University of Essex Library Services.

[58] Hariharan Subramonyam, Wilmot Li, Eytan Adar, and Mira Dontcheva. 2018. TakeToons: Script-driven Performance Animation. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18)*. Association for Computing Machinery, New York, NY, USA, 663–674. https://doi.org/10.1145/3242587.3242618

[59] Hariharan Subramonyam, Colleen Seifert, Priti Shah, and Eytan Adar. 2020. texSketch: Active Diagramming through Pen-and-Ink Annotations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376155

[60] Bernhard Suhm, Brad Myers, and Alex Waibel. 2001. Multimodal error correction for speech user interfaces. *ACM Transactions on Computer-Human Interaction* 8, 1 (March 2001), 60–98. https://doi.org/10.1145/371127.371166

[61] Ivan E. Sutherland. 1963. Sketchpad: a man-machine graphical communication system. In *Proceedings of the May 21-23, 1963, spring joint computer conference (AFIPS '63 (Spring))*. Association for Computing Machinery, New York, NY, USA, 329–346. https://doi.org/10.1145/1461551.1461591

[62] William Robert Sutherland. 1966. *The on-line graphical specification of computer procedures.* Thesis. Massachusetts Institute of Technology. https://dspace.mit.edu/handle/1721.1/13474 Accepted: 2005-09-21T22:40:43Z.

[63] Ryo Suzuki, Rubaiat Habib Kazi, Li-yi Wei, Stephen DiVerdi, Wilmot Li, and Daniel Leithinger. 2020. RealitySketch: Embedding Responsive Graphics and Visualizations in AR through Dynamic Sketching. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST '20)*. Association for Computing Machinery, New York, NY, USA, 166–181. https://doi.org/10.1145/3379337.3415892

[64] Phil Turner. 2016. A Make-Believe Narrative for HCI. In *Digital Make-Believe*, Phil Turner and J. Tuomas Harviainen (Eds.). Springer International Publishing, Cham, 11–26. https://doi.org/10.1007/978-3-319-29553-4_2

[65] Barbara Tversky. 2011. Visualizing Thought. *Topics in Cognitive Science* 3, 3 (2011), 499–535. https://doi.org/10.1111/j.1756-8765.2010.01113.x _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1756-8765.2010.01113.x.

[66] Bret Victor. 2014. Humane representation of thought: a trail map for the 21st century. In *Proceedings of the 27th annual ACM symposium on User interface software and technology (UIST '14)*. Association for Computing Machinery, New York, NY, USA, 699. https://doi.org/10.1145/2642918.2642920

[67] Vihart. 2016. The Case for Hovercars. https://www.youtube.com/watch?v=3zQsYJi5pFE

[68] Terry Winograd. 1972. Understanding natural language. *Cognitive Psychology* 3, 1 (Jan. 1972), 1–191. https://doi.org/10.1016/0010-0285(72)90002-3

[69] Haijun Xia. 2020. Crosspower: Bridging Graphics and Linguistics. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST '20)*. Association for Computing Machinery, New York, NY, USA, 722–734. https://doi.org/10.1145/3379337.3415845

[70] Haijun Xia, Tony Wang, Aditya Gunturu, Peiling Jiang, William Duan, and Xiaoshuo Yao. 2023. CrossTalk: Intelligent Substrates for Language-Oriented Interaction in Video-Based Communication and Collaboration. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3586183.3606773

A FORMATIVE STEPS ADDITIONAL CONTENT

Figure 14 shows a sample of motivating content. Figure 15 shows P1's result from the formative exercises.

B EXPLORATORY STUDY WITH DRAWTALKING — ADDITIONAL CONTENT

B.1 Participants' Backgrounds

The following summarizes the participants' backgrounds:

P1$_{expl}$ (gender: male, age: 46 years): An experienced professor (at a higher-level institution) in computer science, interactive graphics, and new visual media (e.g. AR, VR storytelling and games) — focuses on teaching students new to programming. Uses many visual aids in classes.

P2$_{expl}$ (gender: female, age: 21 years): Digital fine arts university student (design and computer science), drawing is a hobby. Uses sketching for game design; planning gameplay features and figuring out how they should work. e.g. diagramming and framework design. Experienced with drawing on tablets.

P3$_{expl}$ (gender: male, age: 23 years): Experience as a digital designer, economics and digital art university student, *not* professionally-trained in art, uses a digital drawing application (Procreate) to record visual ideas.

P4$_{expl}$ (gender: female, age: 20 years): Tech / Design university student. Does design for web, desktop, apps in-general. Has done digital illustration and created drawings since childhood.

P5$_{expl}$ (gender: male, age: 49 years): An experienced professor in computer programming for interactive graphics. Does live-coding and streaming for internet-based education as well. Uses and authors open-source coding libraries for interactive graphics. Teaches programming for creating visuals rather than designing visuals by hand.

P6$_{expl}$ (gender: female, age: 18 years): Student in interactive media. Draws and paints using physical media (e.g. paintbrush and canvas), with lifelong interest and experience. Experienced with digital graphical design work. Low experience with programming.

P7$_{expl}$ (gender: male, age: 36 years): Industry creative expert. 15+ years experience as a set designer and in the theater industry. Digital illustrator, spatial experience designer, AR/VR immersive projects, projects in entertainment and fine-arts. Creative directing, collaborations with companies and academia. Non-programmer, some experience with visual blocks-based interfaces.

P8$_{expl}$ (gender: female, age: 26 years): Robotics and machine-learning researcher, works with developing ML models, often needs to visualize ML inputs/outputs and illustrates rough sketches to think-through policies (for robots). No prior experience using a tablet.

P9$_{expl}$ (gender: female, age: 24 years): Robotics and machine-learning PhD student, used tablet-based sketching interfaces.

B.2 Select Screenshots

A selection of examples from the user sessions are shown: P1 Figure 16, P2 Figure 17, P3 Figure 18, P4 Figure 2, P5 Figure 19, P6 Figure 20, P7 Figure 21, P8 Figure 22, P9 Figure 23.

C EXPLORATORY FEATURES AND EXAMPLES FOR THE RECORD

During the development of this project, we explored several ideas mixing interactive visuals and text. Some of these did not contribute to the core ideas and this leg of the research was complete without them, but the examples might be useful to include for readers interested in combining more direct-manipulation techniques.

For example: reusable text commands in Figure 24 and context-sensitive representations of numerical outputs based on labels in Figure 25.

We also prototyped an additional working example specializing in "digital interactive assignments" in which the user builds a molecule-matching game. See Figure 26. An empty "box" sketch, a "checkbox" sketch, and a hierarchical "water" sketch have been

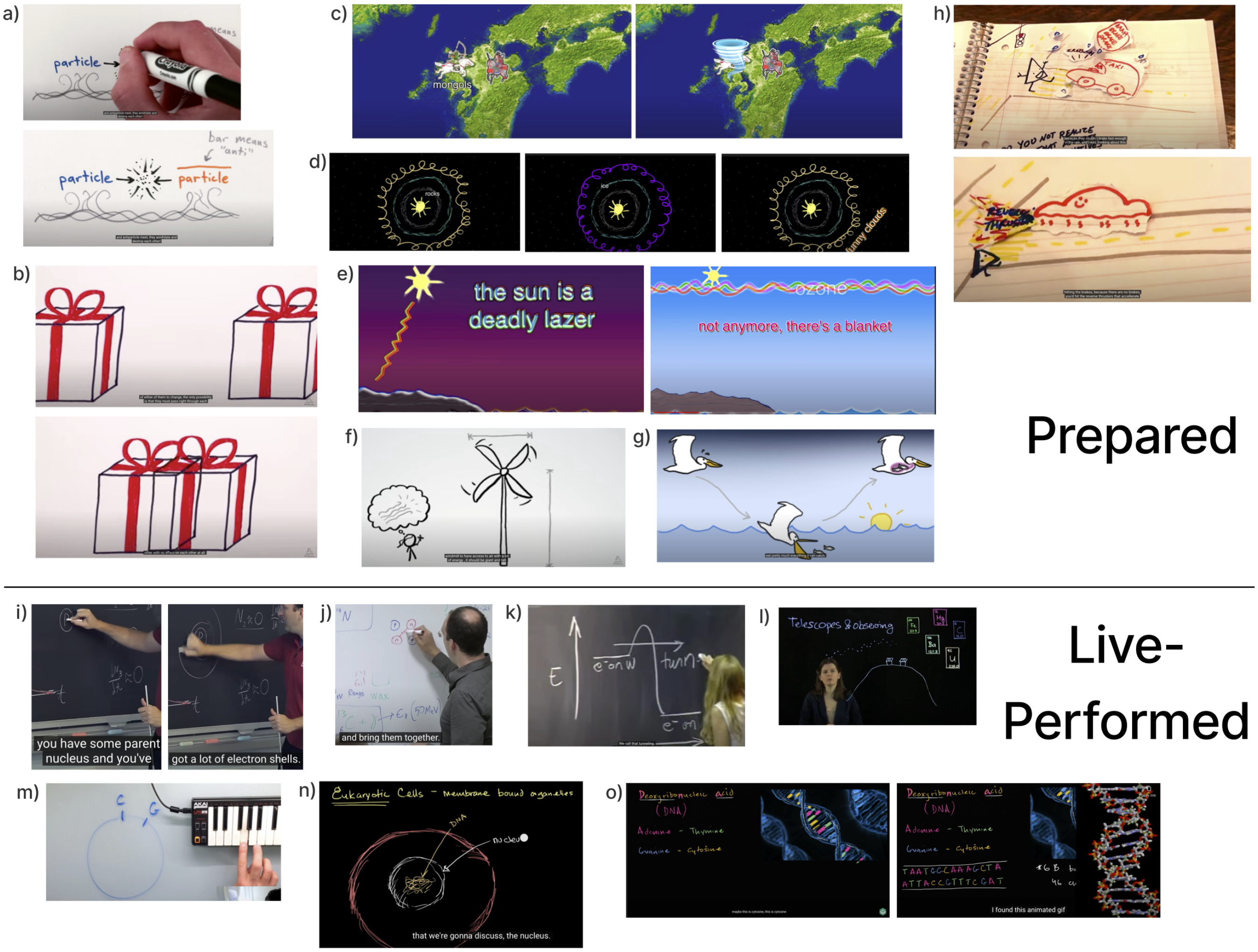


Figure 14: Formative Content Exploration: (top) subset of prepared and (bottom) live-performed sketch-based content

Attribution and Disclaimer: *All rights belong to the original creators of the referenced video content. The screen-shots are for the sole purpose of providing examples of content used as partial inspiration for our research.*

a ©Neptune Studios — minutephysics — Antimatter Explained[39]
b ©Neptune Studios — minutephysics — Immovable Object vs. Unstoppable Force - Which Wins?[38]
c ©Bill Wurtz — history of japan[5]
d, e ©Bill Wurtz — history of the entire world, i guess[6]
f ©Neptune Studios — minutephysics — The Physics of Windmill Design[40]
g ©Neptune Studios — MinuteEarth — Why These Bears "Waste" Food[37]
h ©Vi Hart — The Case for Hovercars[67]
i ©MIT OpenCourseWare — MIT 22.01 Introduction to Nuclear Engineering and Ionizing Radiation, Fall 2016 — 11. Radioactivity and Series Radioactive Decays[43]
j ©MIT OpenCourseWare — MIT 22.01 Introduction to Nuclear Engineering and Ionizing Radiation, Fall 2016 — 1. Radiation History to the Present — Understanding the Discovery of the Neutron[42]
k ©MIT OpenCourseWare — Atomic Theory of Matter — Lec 1 | MIT 5.111 Principles of Chemical Science, Fall 2005[41]
l ©MIT OpenCourseWare — MIT RES.8-007 Cosmic Origin of the Chemical Elements, Fall 2019 — Ep. 6: Element Production (Fusion) – Part 1[44]
m ©Michael New — The Circle of Fifths - How to Actually Use It[35]
n ©Khan Academy — Organelles in eukaryotic cells | The cellular basis of life | High school biology | Khan Academy[21]
o ©Khan Academy — DNA | Biomolecules | MCAT | Khan Academy[20]

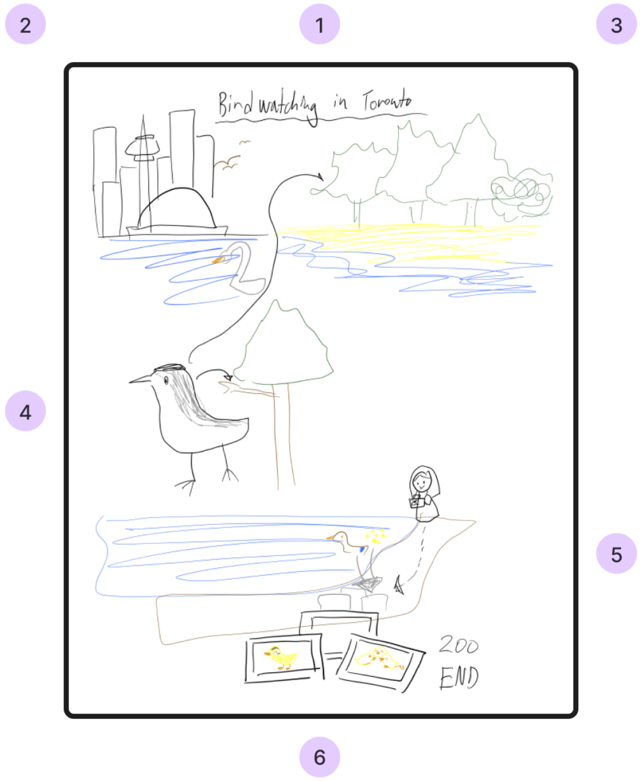
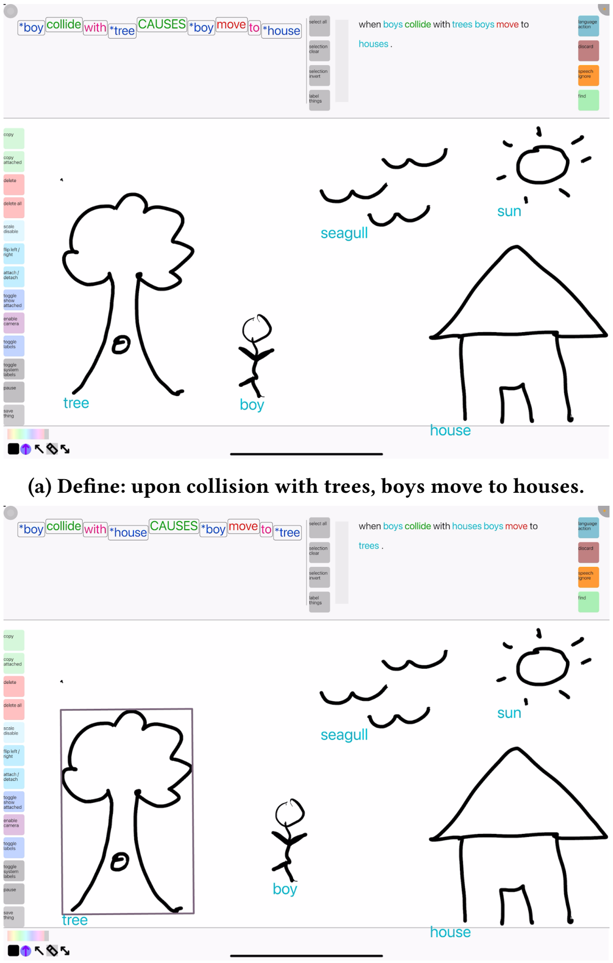


Figure 15: Formative Sketching Session, P1 final image: P1 narrated their own bird-watching story while sketching. Their iterative workflow and visual style of moving between "islands of locations" helped inspire some features: e.g. discrete objects, object hierarchies, and spatially-flexible object relationships. Drawn roughly in-order: 1) Inline text titling the "story"; 2) drawing of the city Toronto to set the location; 3) P1 visited the forest in Toronto; 4) P1 saw a bird, drew arrows indicating object relation (on a tree, in the forest); 5) P1 (the person sketch) found a pond and watched ducklings; 6) P1 took 200 photos.

saved prior that looks like the structure shown, with "water" labeled. (These were deleted prior.) Next, the rule was created: *"When atoms **form** water, the box transforms into a checkbox."*. This creates a script that checks whether the user has created any sketch structure matching the names and hierarchy of the target. We believe the matching functionality of "form" would be useful to have available in an interactive lesson notebook or textbook.

Towards additional exploration of error-tolerance, we implemented an early version of unknown verb substitution for an extension of the semantics diagram Figure 27. Unknown verbs must be substituted in the semantics diagram for the command to proceed. Once the verb is substituted, the mapping persists so as not to interrupt the user again. The idea is to interrupt the user only when necessary (i.e. there is ambiguity.) (The suggestions are found simply using WordNet [36] within NLTK[33].) Context-sensitive error recovery components like this could be explored more deeply.


*boy collide with *tree CAUSES *boy move to *house

when boys collide with trees boys move to houses .

(a) Define: upon collision with trees, boys move to houses.

*boy collide with *house CAUSES *boy move to *tree

when boys collide with houses boys move to trees .

(b) Define: upon collision with houses, boys move to trees.

Figure 16: *P1: Creatively defining a custom loop using rules.* P1 discovered a way to make a kind of infinite loop using 2 rules to cause an infinite sequence of collision/response movements back and forth between the house and the tree.

D ABSTRACT SEMANTIC STRUCTURE

The following defines the basic structure of S2 (section 5.2), which is a simplified semantic structure graph representing an interpretable command. The concrete implementation of DrawTalking traverses this structure to generate final execution commands S3 (section 5.2).

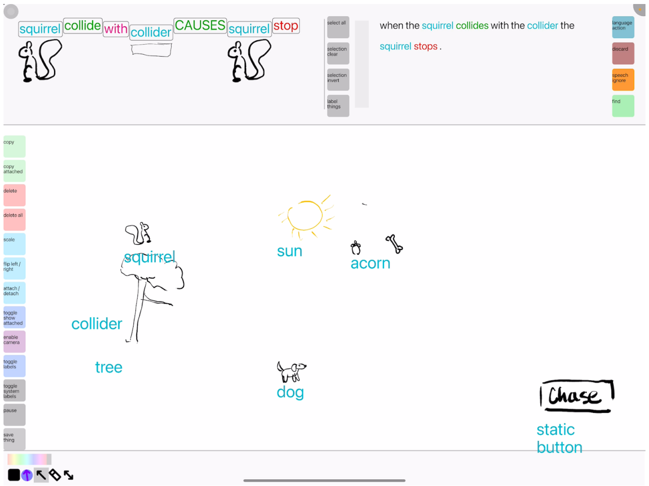

when the squirrel collides with the collider the squirrel stops .

Figure 17: ***P2: Squirrel Climbing Tree Stopped by Collider.*** **A rule is assigned to the collider to make it a prop the user can move around themselves to stop the squirrel from climbing upon collision with the collider. (The collider is shown in the semantics diagram as a rectangle, but invisible in the canvas on-purpose — where the label "collider" is.) P2 also created a button that causes the dog to move.**

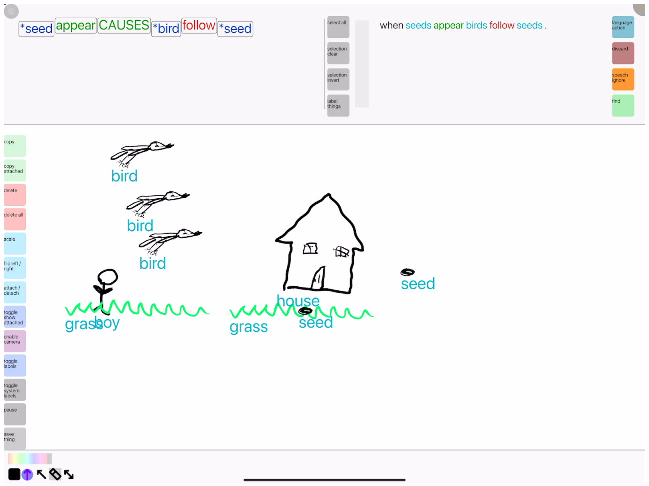

when seeds appear birds follow seeds .

Figure 18: ***P3: "When seeds appear birds follow seeds."***

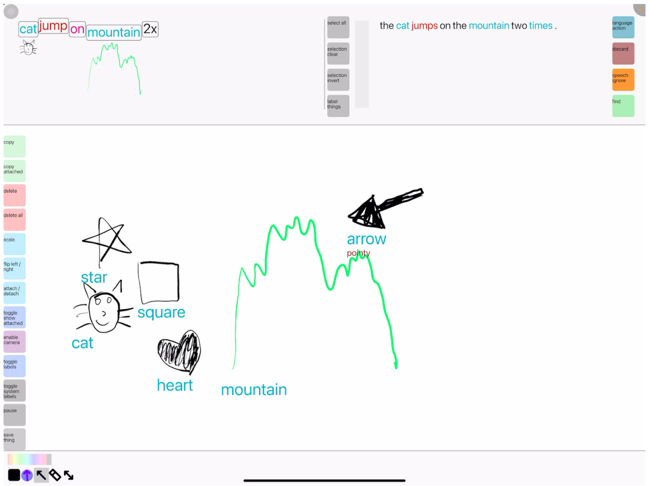

the cat jumps on the mountain two times .

Figure 19: ***P5: "The cat jumps on the mountain two times."***

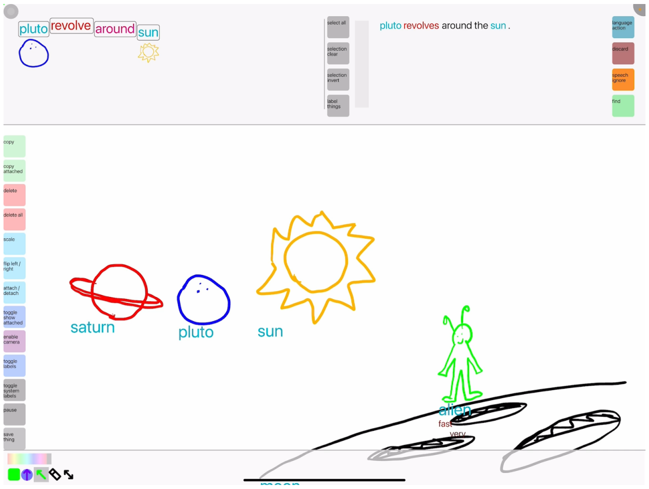
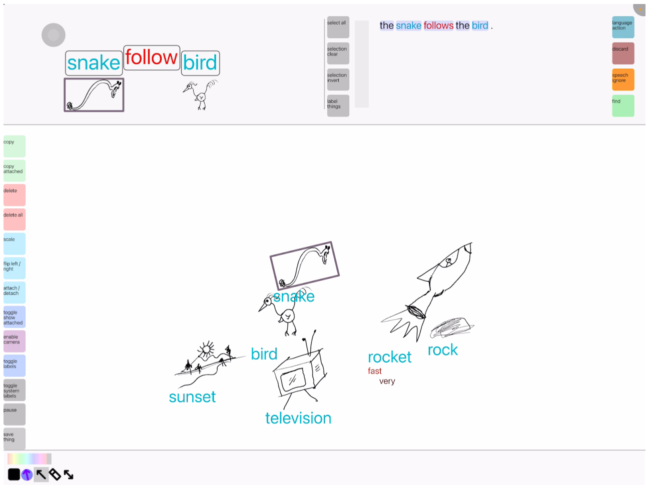

pluto revolves around the sun .

Figure 20: ***P6: "Pluto revolves around the sun."***



the snake follows the bird .

Figure 21: *P7: "The snake follows the bird."*

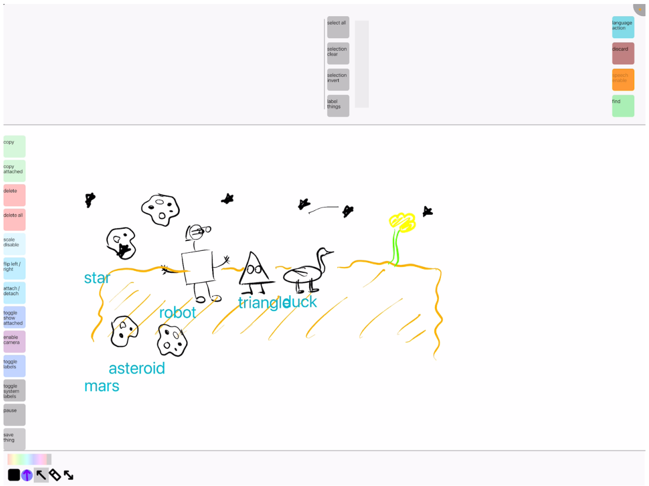
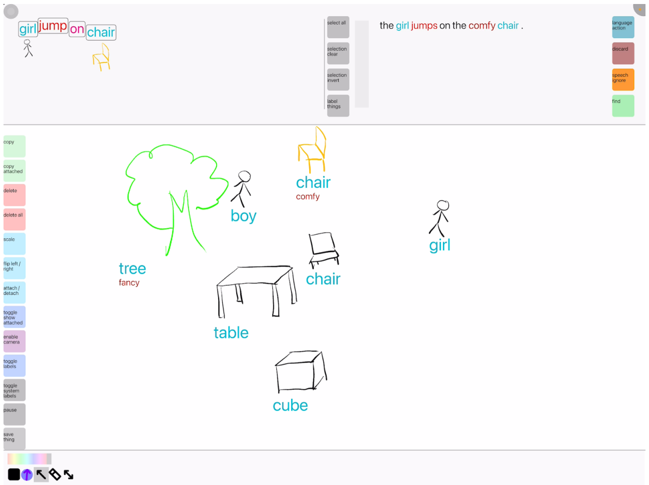
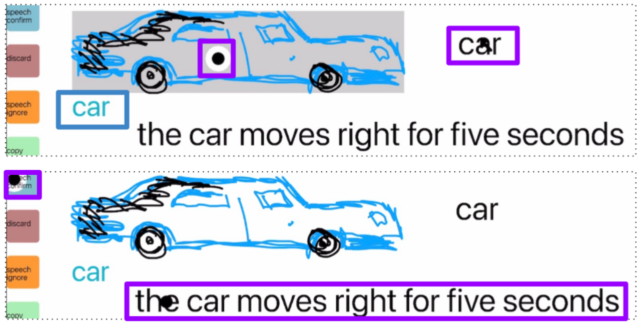


Figure 22: *P8: Space scene with robots, ducks, asteroids, and stars near Mars.*



the girl jumps on the comfy chair .

Figure 23: *P9: "The girl jumps on the comfy chair."*



the car moves right for five seconds

the car moves right for five seconds

Figure 24: Text objects as macros: We can reduce repetitive speech by storing text as inline objects that work exactly as text in the semantics diagram (Figure 4). They're created by linking the text transcript's current content and the canvas. Here, inline text objects (in purple) have been created from 1) the word "car" and 2) the sentence, "the car moves right for five seconds." The user can link a new object with 1) to label it "car" and link 2) and the language action button to treat the text as a reusable command. (Running "the car moves right..." again.) There is potential to explore this further. However, we decided to focus on the more general capability of triggering commands with rules, e.g. upon pressing a button.

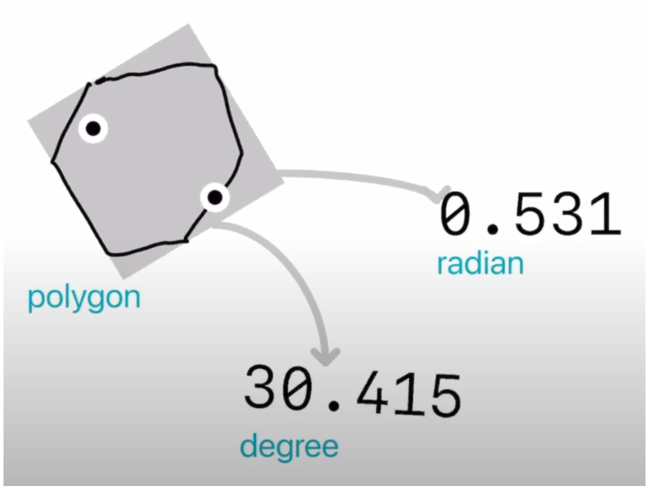


Figure 25: Context-sensitive representations based on labels: We briefly explored using the labels to control the visual representation of "number sketches." In this case, arrows connected from a freehand sketch output the sketch's angle to the numbers, and the labels determine whether to display the angle in degrees or radians. "Dynamic representations" was left to future work.

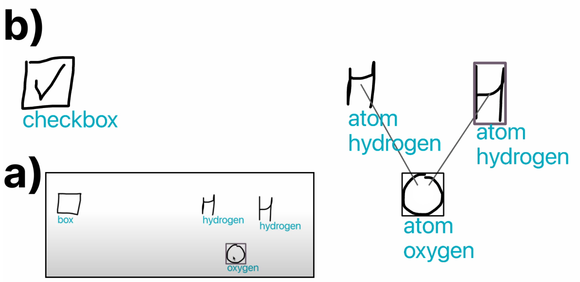


Figure 26: Molecule Matching Lesson prototype: This example shows a checkbox (initially empty: a) that has automatically been filled (b) after the user has recreated the structure of a sketched water (H_2O) molecule formed from sketches labeled atoms, hydrogen, and oxygen.

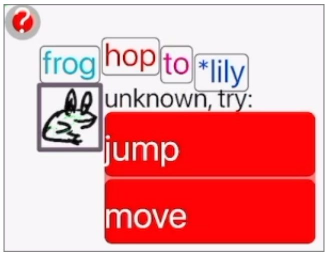


Figure 27: Substitution suggestions for unknown verbs: This extension to the semantics diagram is an early exploration in error-tolerance that could be developed. When unknown verbs are used in a command, the semantics diagram displays a list of potentially-equivalent words from the existing set, for each unknown verb. The user can select one or opt to cancel. Once the verb is substituted, the mapping persists so as not to interrupt the user again. (Here, the command is *"The frog hops to a lily"*, but "hop" is undefined.)

Listing 1: S2 Data Structure

```
S2_Element : $\textbf{Type\_Definition}$ {
    type :=
        CMD_LIST |
        ACTION |
        AGENT | DIRECT_OBJECT | OBJECT | INDIRECT_OBJECT |
        PREPOSITION |
        TRIGGER_RESPONSE | TRIGGER | RESPONSE |
        SEQUENCE_SIMULTANEOUS | SEQUENCE_THEN |
        PROPERTY | PLURAL | COUNT | SPECIFIC_OR_UNSPECIFIC | TIME |
        COREFERENCE

    value :=
        Number | // e.g. any of Float64, Float32, Uint64, etc.
        Thing_ID |
        Thing_Type |
        String |
        Boolean |
        // e.g. pointer or int ID to dynamic-allocated object
        Reference |
        List[Value_Type]

    // S2_Elements should be allocated with stable pointers, or use stable IDs
    parent : Reference(S2_Element)
    // Similar to JSON, but elements are always lists
    // (can have multiple children for the same key
    // (although the layout is not a hard requirement)
    key_to_value := Map[String : List[S2_Element]]
    // property can refer to another property e.g. for coreference
    refers_to : Reference(S2_Element)
    // usually refers to some user feedback UI element
    user_feedback_ref : Reference(Anything)
    // optional: stable pointer or ID to the raw token in
    // the language input used to create this element
    token : Reference(Token)
}
```

Listing 2: Possible structures for S2

```
// Note that each right-hand-side can be a list.

CMD_LIST -> any of the rest

ACTION -> any combination of
         SOURCE,
         DIRECT_OBJECT,
         INDIRECT_OBJECT,
         OBJECT,
         PREPOSITION,
         // contains a "trait" with a string value for the property name
         // e.g. how adverbs or adjectives are used.
         PROPERTY,
         SEQUENCE_SIMULTANEOUS,
         SEQUENCE_THEN,
         // usually how long the action should last
         TIME
         COREFERENCE,

TRIGGER_RESPONSE -> TRIGGER + RESPONSE

TRIGGER, RESPONSE -> ACTION
// equivalent to ACTION.
// (but handled differently to generate rules.
// TRIGGER should be used to generate rules.
// RESPONSE should be used to generate commands
// invoked in the future with arguments generated from rule evaluation).

PREPOSITION -> OBJECT

SOURCE, DIRECT_OBJECT, INDIRECT_OBJECT, OBJECT -> any combination of
PLURAL, // whether plural or not
COUNT, // number of elements
SPECIFIC_OR_UNSPECIFIC // referring to a specific object or not
PROPERTY

PLURAL, COUNT, SPECIFIC_OR_UNSPECIFIC, TIME are terminal

SEQUENCE_THEN -> ACTION
SEQUENCE_SIMULTANEOUS -> ACTION

COREFERENCE // contains a pointer to another node in the value, usually noun-like.
```

Listing 3: Output from "Forever the person throws the ball into the pond and then the dog gives the ball to her." prior to object selection

```
{
    label=[], tag=[], type=[], kind=[], key=[], idx=[0] id=[1931]
    [CMD_LIST] = [
    {
        label=[], tag=[], type=[CMD], kind=[], key=[CMD_LIST], idx=[0] id=[1932]
        [ACTION] = [
        {
            label=[throw], tag=[VERB], type=[ACTION], kind=[ACTION], key=[ACTION], idx=[0] id=[1933]
            [PREPOSITION] = [
            {
                label=[into], tag=[], type=[into], kind=[], key=[PREPOSITION], idx=[0] id=[1934]
                [OBJECT] = [
                {
                    label=[pond], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[OBJECT], idx=[0] id=[1935]
                    value={
                        THING_INSTANCE=[609]
                    }
                    [SPECIFIC_OR_UNSPECIFIC] = [
                    {
                        label=[the], tag=[DET], type=[VALUE], kind=[SPECIFIC], key=[SPECIFIC_OR_UNSPECIFIC], idx=[0] id=[1936]
                        value={
                            FLAG=[true]
                        }
                    }
                    ,
                    ]
                    [COUNT] = [
                    {
                        label=[], tag=[], type=[VALUE], kind=[], key=[COUNT], idx=[0] id=[1937]
                        value={
                            NUMERIC=[1.000000]
                        }
                    }
                    ,
                    ]
                    [PLURAL] = [
                    {
                        label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1938]
                        value={
                            FLAG=[false]
                        }
                    }
                    ,
                    ]
                }
                ,
                ]
            }
            ,
            ]
            [SEQUENCE_THEN] = [
            {
                label=[give], tag=[VERB], type=[ACTION], kind=[ACTION], key=[SEQUENCE_THEN], idx=[0], @=[MUST_FILL_IN_AGENT] id=[1939]
                [PREPOSITION] = [
                {
                    label=[to], tag=[], type=[to], kind=[], key=[PREPOSITION], idx=[0] id=[1940]
                    [OBJECT] = [
                    {
                        label=[person], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[AGENT], idx=[0] id=[1968]
                        <coreference substitution> id=[1960]
                        value={
                            THING_INSTANCE=[606]
                        }
                        [SPECIFIC_OR_UNSPECIFIC] = [
                        {
                            label=[the], tag=[DET], type=[VALUE], kind=[SPECIFIC], key=[SPECIFIC_OR_UNSPECIFIC], idx=[0] id=[1969]
                            value={
                                FLAG=[true]
                            }
                        }
                        ,
                        ]
                        [COUNT] = [
                        {
                            label=[], tag=[], type=[VALUE], kind=[], key=[COUNT], idx=[0] id=[1970]
                            value={
                                NUMERIC=[1.000000]
                            }
                        }
                        ,
                        ]
                        [PLURAL] = [
                        {
                            label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1971]
                            value={
                                FLAG=[false]
                            }
                        }
                        ,
                        ]
                    }
                    ,
                    ]
                }
                ,
                ]
                [DIRECT_OBJECT] = [
                {
                    label=[ball], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[DIRECT_OBJECT], idx=[0] id=[1964]
                    <coreference substitution> id=[1955]
                    value={
                        THING_INSTANCE=[607]
                    }
                    [SPECIFIC_OR_UNSPECIFIC] = [
                    {
                        label=[the], tag=[DET], type=[VALUE], kind=[SPECIFIC], key=[SPECIFIC_OR_UNSPECIFIC], idx=[0] id=[1965]
                        value={
                            FLAG=[true]
                        }
                    }
                    ,
                    ]
                    [COUNT] = [
                    {
                        label=[], tag=[], type=[VALUE], kind=[], key=[COUNT], idx=[0] id=[1966]
                        value={
                            NUMERIC=[1.000000]
                        }
                    }
                    ,
                    ]
                    [PLURAL] = [
                    {
                        label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1967]
                        value={
                            FLAG=[false]
                        }
                    }
                    ,
                    ]
                }
                ,
                ]
                [AGENT] = [
                {
                    label=[dog], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[AGENT], idx=[0] id
                            ↪ =[1951]
                    value={
                        THING_INSTANCE=[608]
                    }
                    [SPECIFIC_OR_UNSPECIFIC] = [
                    {
                        label=[the], tag=[DET], type=[VALUE], kind=[SPECIFIC], key=[
                                ↪ SPECIFIC_OR_UNSPECIFIC], idx=[0] id=[1952]
                        value={
                            FLAG=[true]
                        }
                    }
                    ,
                    ]
                    [COUNT] = [
                    {
                        label=[], tag=[], type=[VALUE], kind=[], key=[COUNT], idx=[0] id=[1953]
                        value={
                            NUMERIC=[1.000000]
                        }
                    }
                    ,
                    ]
                    [PLURAL] = [
                    {
                        label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1954]
                        value={
                            FLAG=[false]
                        }
                    }
                    ,
                    ]
                }
                ,
                ]
            }
            ,
            ]
            [DIRECT_OBJECT] = [
            {
                label=[ball], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[DIRECT_OBJECT], idx=[0] id
                        ↪ =[1955]
                value={
                    THING_INSTANCE=[607]
                }
                [SPECIFIC_OR_UNSPECIFIC] = [
                {
                    label=[the], tag=[DET], type=[VALUE], kind=[SPECIFIC], key=[SPECIFIC_OR_UNSPECIFIC],
                            ↪ idx=[0] id=[1956]
                    value={
                        FLAG=[true]
                    }
                }
                ,
                ]
                [COUNT] = [
                {
                    label=[], tag=[], type=[VALUE], kind=[], key=[COUNT], idx=[0] id=[1957]
                    value={
                        NUMERIC=[1.000000]
                    }
                }
                ,
                ]
                [PLURAL] = [
                {
                    label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1958]
                    value={
                        FLAG=[false]
                    }
                }
                ,
                ]
            }
```

```
            ,
            ]
            [PROPERTY] = [
            {
                label=[modifier], tag=[ADV], type=[PROPERTY], kind=[], key=[PROPERTY], idx=[0] id=[1959]
                value={
                    TEXT=[forever]
                }
            }
            ,
            ]
            [AGENT] = [
            {
                label=[person], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[AGENT], idx=[0] id=[1960]
                value={
                    THING_INSTANCE=[606]
                }
                [SPECIFIC_OR_UNSPECIFIC] = [
                {
                    label=[the], tag=[DET], type=[VALUE], kind=[SPECIFIC], key=[SPECIFIC_OR_UNSPECIFIC],
                            ↪ idx=[0] id=[1961]
                    value={
                        FLAG=[true]
                    }
                }
                ,
                ]
                [COUNT] = [
                {
                    label=[], tag=[], type=[VALUE], kind=[], key=[COUNT], idx=[0] id=[1962]
                    value={
                        NUMERIC=[1.000000]
                    }
                }
                ,
                ]
                [PLURAL] = [
                {
                    label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1963]
                    value={
                        FLAG=[false]
                    }
                }
                ,
                ]
            }
            ,
            ]
        }
        ,
        ]
    }
    ,
    ]
}
```

Listing 4: Output from "Every few seconds the frog hops to a lily." prior to object selection

```
{
    label=[], tag=[], type=[], kind=[], key=[], idx=[0] id=[1018]
    [CMD_LIST] = [
    {
        label=[], tag=[], type=[CMD], kind=[], key=[CMD_LIST], idx=[0] id=[1019]
        [ACTION] = [
        {
            label=[hop], tag=[VERB], type=[ACTION], kind=[ACTION], key=[ACTION], idx=[0] id=[1005]
            [PREPOSITION] = [
            {
                label=[to], tag=[], type=[to], kind=[], key=[PREPOSITION], idx=[0] id=[1013]
                [OBJECT] = [
                {
                    label=[lily], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[OBJECT], idx=[0] id
                            ↪ =[1014]
                    value={
                        THING_INSTANCE=[0]
                    }
                    [SPECIFIC_OR_UNSPECIFIC] = [
                    {
                        label=[a], tag=[DET], type=[VALUE], kind=[UNSPECIFIC], key=[
                                ↪ SPECIFIC_OR_UNSPECIFIC], idx=[0] id=[1016]
                        value={
                            FLAG=[false]
                        }
                    }
                    ,
                    ]
                    [COUNT] = [
                    {
                        label=[], tag=[], type=[], kind=[], key=[COUNT], idx=[0] id=[1017]
                        value={
                            NUMERIC=[1.000000]
                        }
                    }
                    ,
                    ]
                    [PLURAL] = [
                    {
                        label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1015]
                        value={
                            FLAG=[false]
                        }
                    }
                    ,
                    ]
                }
                ,
                ]
            }
            ,
            ]
            [TIME] = [
            {
                label=[second], tag=[TIME], type=[INTERVAL], kind=[], key=[TIME], idx=[0] id=[1006]
                [PROPERTY] = [
                {
                    label=[trait], tag=[ADJ], type=[PROPERTY], kind=[], key=[PROPERTY], idx=[0] id=[1008]
                    value={
                        TEXT=[few]
                    }
                }
                ,
                ]
            }
            ,
            ]
            [AGENT] = [
            {
                label=[frog], tag=[NOUN], type=[], kind=[THING_INSTANCE], key=[AGENT], idx=[0] id=[1009]
                value={
                    THING_INSTANCE=[0]
                }
                [SPECIFIC_OR_UNSPECIFIC] = [
                {
                    label=[the], tag=[DET], type=[VALUE], kind=[SPECIFIC], key=[SPECIFIC_OR_UNSPECIFIC],
                        ↪ idx=[0] id=[1011]
                    value={
                        FLAG=[true]
                    }
                }
                ,
                ]
                [COUNT] = [
                {
                    label=[], tag=[], type=[VALUE], kind=[], key=[COUNT], idx=[0] id=[1012]
                    value={
                        NUMERIC=[1.000000]
                    }
                }
                ,
                ]
                [PLURAL] = [
                {
                    label=[], tag=[], type=[VALUE], kind=[], key=[PLURAL], idx=[0] id=[1010]
                    value={
                        FLAG=[false]
                    }
                }
                ,
                ]
            }
            ,
            ]
        }
        ,
        ]
    }
    ,
    ]
}
```