AuthorIVE: Authoring Interactions for Virtual Environments through Disambiguating Demonstrations

Stephanie Claudino Daffara
Federico Saldarini
Balasaravanan Thoravi Kumaravel
Björn Hartmann

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-122
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-122.html

May 29, 2020

Acknowledgement

I would like to thank my advisor, Professor Björn Hartmann, for guiding and mentoring me along the way. I would also like to thank Bala Kumaravel for his dedication, hard work, and invaluable advise, and Fede Saldarini, without whom this work would not have been possible. Thank you Fede for your technical and emotional endless support throughout my entire academic career. Finally, I'd like to thank my family and friends for their unconditional love and encouragement.

AuthorIVE: Authoring Interactions for Virtual Environments through Disambiguating Demonstrations

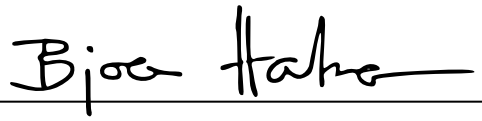
by Stephanie Claudino Daffara

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

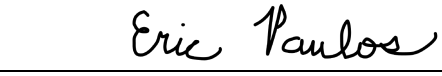


Björn Hartmann
Research Advisor

5/28/2020

(Date)

* * * * * * *



Eric Paulos
Second Reader

28 May 2020

(Date)

AuthorIVE: Authoring Interactions for Virtual Environments through Disambiguating Demonstrations

Stephanie Claudino Daffara*
University of California, Berkeley
Berkeley, California
stephaniecd@berkeley.edu

Federico Saldarini*
San Francisco, California
fede.saldarini@gmail.com

Balasaravanan Thoravi Kumaravel
University of California, Berkeley
Berkeley, California
tkbala@berkeley.edu

Bjoern Hartmann
University of California, Berkeley
Berkeley, California
bjoern@berkeley.edu

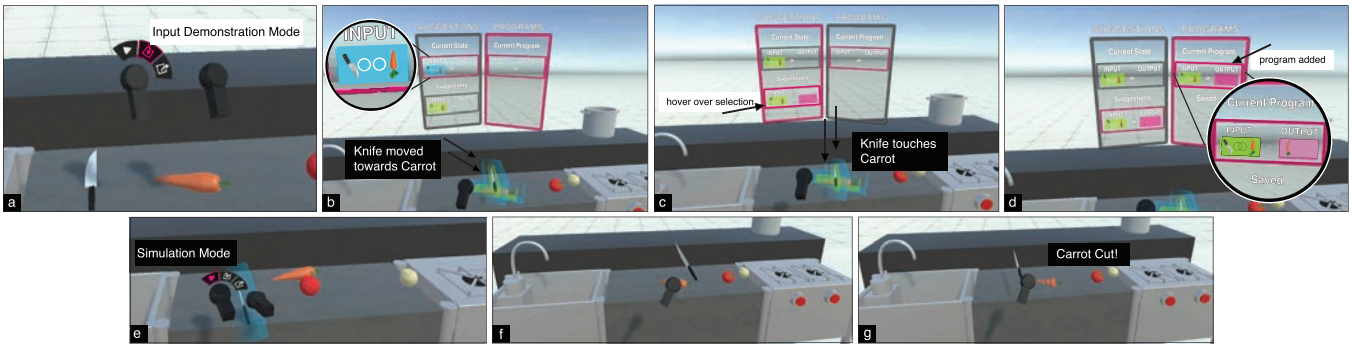


Figure 1: Workflow for authoring the interaction between a Knife and a Carrot using AuthorIVE. (*a*) User switches to Input Demonstration mode. (*b*) User moves the Knife towards the Carrot and gets Current State feedback in the Suggestions Gallery. (*c*) Once the Knife touches the Carrot, the event gets reflected in the Suggestions Gallery and a new suggestion shows up. (*d*) If the user selects the hovered option in (*c*), where if the Knife and Carrot are in contact, the Carrot gets cut, it gets added to the Programs Gallery under the Current Program section. (*e*) User switches to Simulation Mode. (*f, g*) User moves the Knife towards the Carrot and the Carrot gets cut. (Note: UI elements emphasized for clarity)

ABSTRACT

While Virtual Reality (VR) hardware is increasingly available and 3D model repositories are plentiful, creating immersive interactive experiences is still hard for non-programmers. Current content creation tools require reasoning about complex spatial interaction logic in some formal language (e.g., textual or visual programming). Programming by demonstration (PbD) has been a promising direction to allow systems to infer rules from examples. But gaps exist between algorithmic advances in PbD and appropriate user interfaces that enable authors to leverage PbD inference while remaining in control of the authoring process. We contribute a new hybrid authoring method - "disambiguating demonstrations", in which users demonstrate interactions between objects in a VR environment and the system generates suggestions based on these interactions providing resolution of different options along the way. This enables users to create interactive experiences through demonstration. We evaluate our tool through recreating interactive experiences found in other prior work and other existing systems.

CCS CONCEPTS

• **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

Virtual Reality; immersive authoring, disambiguating demonstrations

1 INTRODUCTION

VR is currently used in many different domains, including in games, movies and art and there exists an equally vast number of tools to create content for these domains. Existing tools range from simple template-based applications (such as Oculus' home templates [19]) where users select objects or actions from a predefined set of options, to complex game engines (like Unity3D [26] and Unreal Engine [27]) and CAD-like authoring applications [18].

The simpler systems however, are usually significantly constrained in terms of the interactions that users can author. At the same time, the more flexible tools usually require programming in a formal language, which restricts the tool's user base to individuals familiar with programming.

*Both authors contributed equally to the paper.

In contrast, a tool based on *Programming by Demonstration* (PbD) could be as powerful as the most complex tools while hiding away their complexity by means of an interface that is more accessible to a larger number of users. This is because it only requires that users demonstrate a desired behaviour by the system, through manually demonstrating it themselves.

We introduce AuthorIVE, an **Author**ing tool for creating **I**nteractive **V**irtual **E**nvironments without the use of any formal programming. Our method draws on PbD techniques and interleaves those with choices through a new approach that we call *disambiguating demonstrations*. This technique makes use of two main visualization concepts: Feedback Visualization, a visual representation of the world's current state; and Feedforward Visualization, a visual representation of possible future states which the user can reach from the current state, depending on how they act next.

We hypothesize that interleaving demonstrations with the selection of possible learned rules has the following benefits:

- Reducing the number of demonstrations required to author a rule.
- Reducing the error in rules inferred by the system.
- Encouraging user demonstrations that are less ambiguous to the system and meaningful to the user.
- Providing users with a mental model on what possible rules can be authored with the system.

The real-time aspect of our visualizations in response to literal user actions (a form of immersive authoring [12]) makes our approach particularly well-suited to VR, where users' interactions with a system are inherently more similar to acting or demonstrating than when interacting through 2D interfaces. We achieve this through our implementation of a Suggestions Gallery and a Programs Gallery, both inspired by Marks et al.'s Design Gallery [14] style visualization.

We evaluate AuthorIVE's representational power and flexibility by re-creating interactive environments present in prior work and popular VR applications. We then analyze the environments created through AuthorIVE and use these as bases to discuss AuthorIVE's strengths and limitations.

2 RELATED WORK

2.1 Programming by Demonstration

Programming by Demonstration [4] is a well-studied paradigm with proven applications to domains such as spreadsheets [7], text processing [29], and robotics [30]. It is also popularly used in SVG programming [8], procedural tutorials [11, 13] and games [15]. In PbD, the assumption is that users themselves may not be aware of the underlying logic required to implement a particular behavior or might not have the expertise to express it in a formal way. In such a scenario, users can opt to *Program by Demonstration*, where they demonstrate desired behaviours and actions and the system then figures out the underlying logic required to carry these out.

Previous PbD techniques adopt a staged approach in which demonstrations are performed first and rules are inferred afterward. However, users typically lose agency over the rules learned by these systems during the demonstration phase. Instead, rules are displayed to users post hoc, at which point they can accept, reject and sometimes modify them. But these rules are not explicitly visualized to the user during their demonstration.

Still, PbD removes the need for users to explicitly reason about and specify logical constructs and infers it from user's demonstrations. Mayer and Kuncak [15] explored using PbD to create 2D physics games. While their demonstrations primarily occur through direct manipulation of desired behaviours, their interactions are based on and build over 2D physics simulations. Though Mayer and Kuncak's system automatically synthesizes the code for interaction rules, it still requires users to understand code in order to edit it. A user could also play back the rule and through viewing the event decide if the program was correct or not. In contrast, AuthorIVE focuses on interaction techniques for a 3D environment and does not require users to understand code, let alone synthesize it. While directly editing the code can be a way to exactly specify the rule a user needs, Mayer et. al [16] propose two interaction models for disambiguation program rules: 1) Selection from a list of candidate rules and 2) Active learning from the user through questions to the user. AuthorIVE builds on the former interaction model, and extends similar prior works done in a 2D domain [24, 25] to a 3D environment with an interaction scheme that leverages feedforward guidance for the disambiguation.

2.2 Feedback and Feedforward guidance

Feedforward guidance relates to guiding a user towards performing from a set of all possible actions that can be performed from the current starting state of that user. This can be thought of as providing "paths" the user can possibly take. Whereas in feedback-based guidance, the system evaluates and indicates performance of the user's actions till that instant, providing them with a mental model. Prior works such as Octopocus [2] and Hierarchical marking menus [10] have demonstrated the value of combining these two as a way to train users with an interface, and get them from being a novice to an expert. More fundamentally, the combination makes the user aware of their current state as perceived by the computer system, and shows future actions that could lead to a desired state amongst many potentially reachable states.

Our approach draws from Octopocus, which specifically helps users learn, execute and remember gesture sets, and presents users with this guidance that gets updated in real-time during the course of a demonstration, in response to users' actions. Although a system like Octopocus requires predefined paths that a user can traverse. Still, AuthorIVE leverages this benefit to make the user aware of and reach a desired program. We carry this out within our Suggestions and Programs Gallery based off of Marks et al.'s Design Gallery [14]. Marks et al. presents the user with a set of selectable items that get automatically generated and organized, and contains different graphics or animations per cell that get produced by a varying input or parameter. We organize our feedforward and feedback guidance by using this sort of graphical arrangement within our galleries.

2.3 Authoring Interactive Immersive Environments

Prior works have introduced different techniques for authoring interactivity in immersive environments, which range across different levels of programming expertise that are required. Amongst

these, many works rely on visual scripting approaches that impose reduced requirements on programming expertise. These works still require users to think in terms of explicit logical constructs that are expressed in terms of logic graphs, such as in Ivy [5], Blueprints [23] or in terms of logic blocks as in Alice [9]. Finally through Direct Manipulation as in work by Saquib et. al [21]. MagicalHands [1] also uses direct gestural manipulations to author and modify animation effects of an object, but does not focus on specifying interaction between objects. On the other hand AuthorIVE achieves both the power of expressing logical constructs without the need of a formal language, and the direct interaction logic between objects in a scene.

3 AUTHORIVE

AuthorIVE allows users to author virtual interactions between virtual objects by creating and combining programs, which are simple rules similar to *if-this-then-that* statements. When multiple programs are combined however, they can produce complex behavior.

Users create these programs in a visual and iterative fashion, by cycling between demonstrating interactions and testing them in a simulation environment.

3.1 Modes

Creating a program involves cycling through three modes:

(1) Input Demonstration: where a program's input state is demonstrated.
(2) Output Demonstration: where a program's output state is demonstrated.
(3) Simulation: where the user can directly affect the virtual world to test a program. In this mode the program's input state is checked against the world's state and if these match, the program's output state is executed.

In order to switch between modes users use the *joystick* on their left controller as seen in Figure 2a.

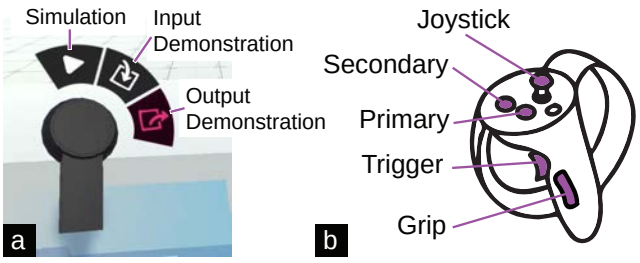


Figure 2: (*a*) Modes menu in VR. (*b*) Oculus touch controller mapping.

3.2 Programs, States and Events

A program is made up of an input state and an output state. States in turn are collections of events and events are boolean predicates involving two objects or an object and an action. Events correspond to three types of occurrences (or lack of occurrences if negated): influence, contact, and action.

Influence and *contact* events are kinematic in nature and are generated whenever a pair of objects is in proximity of each other or in contact with each other. In order to register these kinematic events, each object has an influence bounding volume (shown in Figure 3 as a semi-transparent blue cuboid), and a tighter contact bounding volume (also shown in Figure 3 as a semi-transparent green cuboid). Whenever an object's volumes overlap another object's volumes, events of these types are generated.

Action events are generated whenever an action, such as a tomato being Cut, or balloon Popping, is enabled or disabled. Actions can be enabled or disabled directly by the user or programmatically as part of the execution of the output state of a program.

```
Program:
    InputState:
        (bool, obj_i, obj_j),
        (bool, obj_i, obj_k),
        ...

    OutputState:
        (bool, obj_j, action_k),
        ...
```

Listing 1: Structure of a Program

During Input Demonstration and Output Demonstration modes, the concepts of programs, states and events are conveyed to the user by means of different visual aids:

(1) Bounding Volumes for Kinematic Events: when users hover or interact with objects in the world, we display their bounding volumes in order to explicitly visualize any overlaps which would trigger influence or contact events, as seen in Figure 3.
(2) Suggestions Gallery: As users manipulate objects and change the world's state, we depict the current state (feedback) as well as suggestions based off the current state (feedforward) in a gallery of miniature objects (as seen in Figure 4a and Figure 4b respectively), where action events are rendered explicitly and kinematic events are conveyed through color coding consistent with the bounding volumes. Suggestions can range from partial programs with either only input or output states to full programs, which contain both.
(3) Programs Gallery: As users incrementally build and delete programs, these are shown in their own gallery with the same visual treatment as the Suggestions Gallery (see Figure 4c).

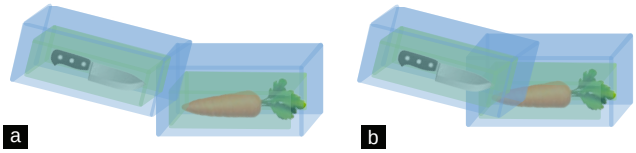


Figure 3: Illustration of the object's bounding volumes. (*a*) Here the influence bounding volumes in translucent blue are overlapping, triggering an influence event. (*b*) Here the Contact bounding volumes in translucent green are overlapping, triggering a contact event.

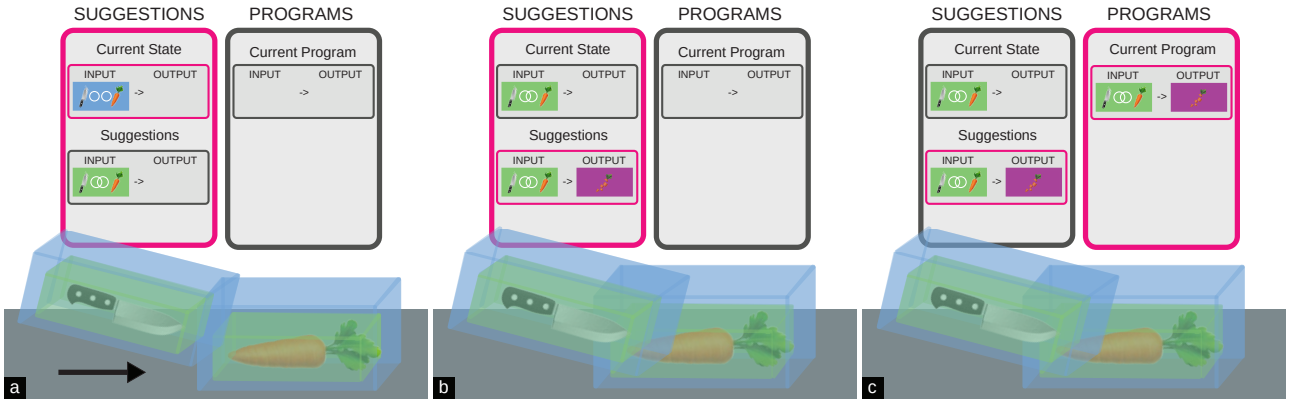


Figure 4: Graphical illustration of the Suggestions and Programs Gallery. (*a*) User overlaps the Knife's and Carrot's influence bounds. (*b*) Contact bounding volumes are now overlapping. User uses the joystick to hover over the first suggestion. (*c*) User has selected the suggestion and it got added to the current program.

3.3 Suggestions and Programs Gallery

AuthorIVE's two galleries work together to disambiguate the user's actions and guide them through the process of selecting input and output events which make up program states.

The Suggestions Gallery provides feedback to the user about the current world state in response to their actions and, guided by a heuristics system, also presents them with partial and complete programs which could be authored given the current state. Users can then choose any of the system's suggestions, which are subsequently merged into the program currently being authored in the Programs Gallery and the process repeats.

The right controller allows interacting with the galleries. The user can move left using the *joystick* (refer to Figure 2b) to select the Suggestions Gallery and right to select the Programs Gallery. If they move up and down, they can scroll through the options of whichever gallery is currently selected. The selected gallery gets a pink highlight (as seen in the Figure 4b versus Figure 4c).

The Suggestion Gallery gets populated with the partial or full completion of a program that a user might want to author. Within it, the *primary button* selects the highlighted input/output events and adds it to the current program within the Programs Gallery (as seen in Figure 4b, and 4c). In the Programs Gallery, the *primary button* saves the current program, and adds it to the saved programs which pops up below the current one. To delete a saved program or clear the current program, with the *joystick* the user can navigate up and down to the desired program and hit the *secondary button*. AuthorIVE always has a current program being authored, therefore if a user saves or clears the current program, or deletes all programs, a new blank current program is created.

The visual language within the gallery of the programs keeps consistent with that of the virtual environment. This helps users visualize the rules and how they might act out in the virtual environment, rather than thinking about them in terms of complex logical constructs.

There are three main visual cues to quickly guide the user towards their intention: Miniature objects, icons and a color scheme. The miniature objects are 3D copies of the virtual object in the scene in their current state. For example, if the Carrot is in a *Cut* state, then it will show up cut such as in Figure 4b in the highlighted suggestion's output. The icons that show up in the input states (seen in Figure 4) each represent a certain kinematic event described in Figure 5. The color scheme follows that of the influence and contact bounding volumes: blue for influence and green for contact.

Figure 5: Kinematic event gallery visuals: (*a*) influence, (*b*) not influence, (*c*) contact, (*c*) not contact.

Since the programs gallery displays what is currently being programmed and all past created programs, it remains visible at all times within demonstration modes. On the other hand the Suggestions Gallery shows up only when hovering over an object or when a held object is interacting with another one.

The galleries give users the power to disambiguate their demonstrations across the different interpreted programs, and does so while the demonstration is ongoing.

4 USING AUTHORIVE

Figure 4 shows a graphical illustration of what authoring the program of "when the Knife gets in contact with the Carrot then perform the Carrot's Cut action" would look like. In Figure 1 we enact the same scenario using the system itself.

For example, in Figure 1a the user starts off by switching into input demonstration mode. Next they move the Knife object near the Carrot object. As the Knife gets closer to the Carrot, it enters the Carrot's influence bounds and the blue influence cuboid and green contact cuboid are revealed (Figure 1b). As this all happens the Suggestions Gallery displays the current state:

```
Current State:
    IN:<influence.Knife.Carrot>,
    OUT:<empty>.
```

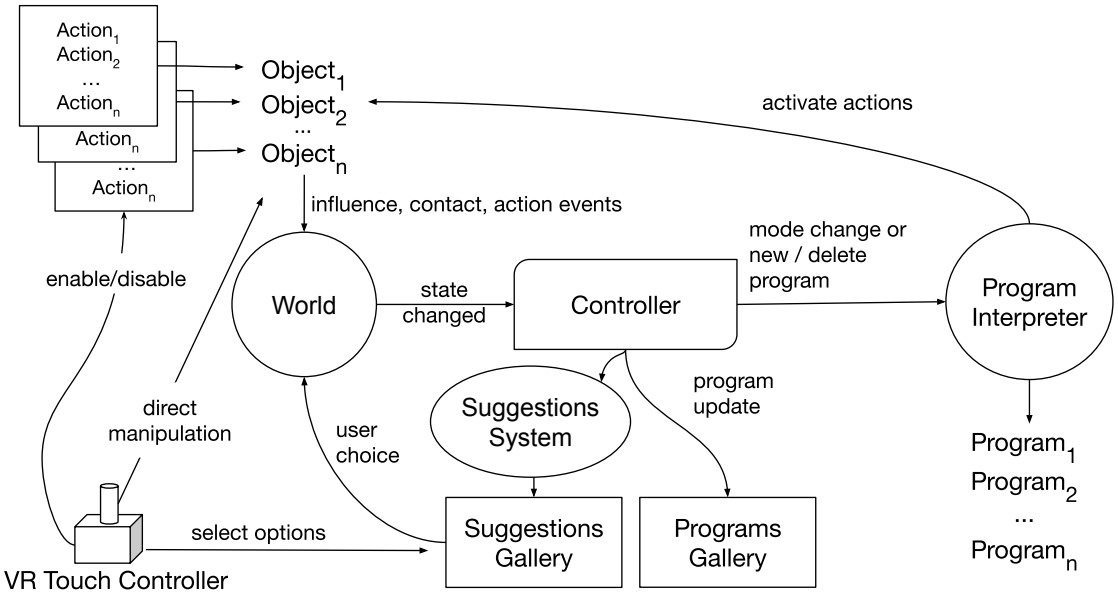


Figure 6: System Diagram. The Controller mediates between the 3 main components: The World, The Suggestions System, and the Program Interpreter.

and suggests that the Knife could get in contact with the Carrot:

```
Suggestions:
    IN:<contact.Knife.Carrot>,
    OUT:<empty>.
```

At all moments the user has the option to either select a suggestion and add that to their current program, or enact the suggestion itself (such as physically moving the Knife closer to the Carrot until they are in contact with each other). In this case the user decides to continue moving the Knife towards the Carrot. Once the objects' contact volumes overlap (Figure 1c), the current state becomes:

```
Current State:
    IN:<contact.Knife.Carrot>,
    OUT:<empty>
```

and the suggestion becomes:

```
Suggestions:
    IN:<contact.Knife.Carrot>,
    OUT:<action.Carrot.Cut>.
```

Since the suggestion has the full program that the user wants, they scroll down with the *joystick* and select the suggestion given. The selected program is then merged with any current program in the Programs Gallery (Figure 1d). This means that if there were already other inputs or outputs within the current program, this new program added gets merged with the preexisting ones.

To test out how that program looks and feels, the user then switches to simulation mode (Figure 1e), and moves the Knife into contact with the Carrot (Figure 1f). As seen in Figure 1g the Carrot gets Cut, which is in line with our authored program.

In this example the use of disambiguation within the galleries might seem simple, but once more objects and actions get added to the scene, there are more opportunities for the system to disambiguate the user's intent. An example of this scenario is depicted in Figure 8.

5 IMPLEMENTATION DETAILS

AuthorIVE is currently implemented in the Unity game engine version 2019.3.7f1 [26] and can be broken down into 3 main implementation components and a central Controller which mediates between these and implements the system as a whole (see Figure 6).

(1) *The World* is responsible for keeping the current state up to date as well as notifying the Controller whenever this state changes. During the course of demonstrations and simulations, objects in the world generate kinematic and action events in response to users' actions. The world coalesces these events into the current state and notifies of any changes with respect to the previous state.
(2) *The Program Interpreter* keeps a list of active programs and, in Simulation mode, evaluates these. On each engine tick, it iterates over all programs and checks if their input state is a subset of the current world state. If this is the case, the conditions for executing the output of a program are considered met and the *interpreter* iterates over all action events in the program's output state, enabling or disabling the corresponding actions for different objects.
(3) *The Suggestions System* is responsible for generating suggestions based on users' demonstrations. When the Controller is in Input Demonstration or Output Demonstration mode, it treats the process of generating suggestions as an online informed search over a (possibly infinite) graph, where the current World state is the current node and there are implicit edges from the current state to possible future World states.

In this context, the *Suggestions System* implements different expansion strategies to decide which nodes to expand next.

The Suggestions System can switch strategies depending on context and can be extended to implement additional strategies, which makes the system particularly flexible.

Our current implementation uses separate strategies for Input and Output Demonstration modes. In Input Demonstration mode we generate suggestions based on a number of heuristics and features:

(1) The concept of an *Instigator* object (the object currently being manipulated). Suggestions are only given in relation to instigator objects.
(2) Increasing distance of other objects to the *Instigator*. And direction of movement (towards or away from the other object). This rule can be visualized in Figure 7
(3) Semantic relationships between the Instigator and nearby objects (i.e.: a knife can cut “cuttable" things). This information is currently provided by the author.

.

For Output Demonstration mode we opt for a simpler heuristic of only suggesting actions for the Instigator object. This is done because the system does not move objects directly (other than through animations the object might already have), the user has to manually move them around. This allows for more real-to-life virtual experiences where an object can be moved and interact with other objects, but the objects do not move around on their own.

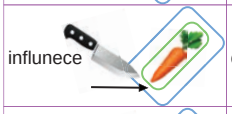
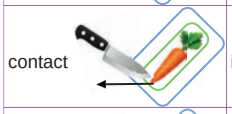
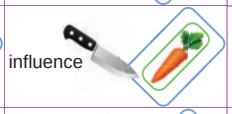
Previous State	Current State	System Suggests
		contact
		nothing
		not in contact
		not in influence

Figure 7: Table of heuristics that dictates what kinematic suggestions gets suggested based on the previous and current states of the world.

6 EVALUATION

To evaluate the extent of interaction environments that AuthorIVE is able to create we conducted four case studies. We used common tools for 3D modeling and scene building (Blender3D [3] and Unity [26]).

6.1 Case Study 1: The Kitchen

In our first case we present the kitchen scene that has been referenced throughout this paper. Here though we exemplify a more complicated scenario. The user has programmed that if the Tomato gets placed on the Fire, the Fire Alarm should sound. And, if the Tomato and Onion get placed in the Pot, and the Pot has water and the stove’s fire is turned on, then the Recipe gets “marked" as Done.

```
Programs:
    Program 1:
        IN:[<contact.Stove.Tomato>,
            <action.Stove.FireOn>],
        OUT:<action.FireAlarm.On>
    Program 2:
        IN:[<contact.Tomato.Pot>,
            <contact.Onion.Pot>,
            <action.Pot.FilledWithWater>,
            <action.Stove.FireOn>],
        OUT:<action.Recipe.MarkDone>
```

This entire sequence can be seen in Figure 8. Although the format of the two programs above is similar to what the *interpreter* uses to execute programs during simulation mode, the user never interfaces with this formalism. Instead the user only interacts with

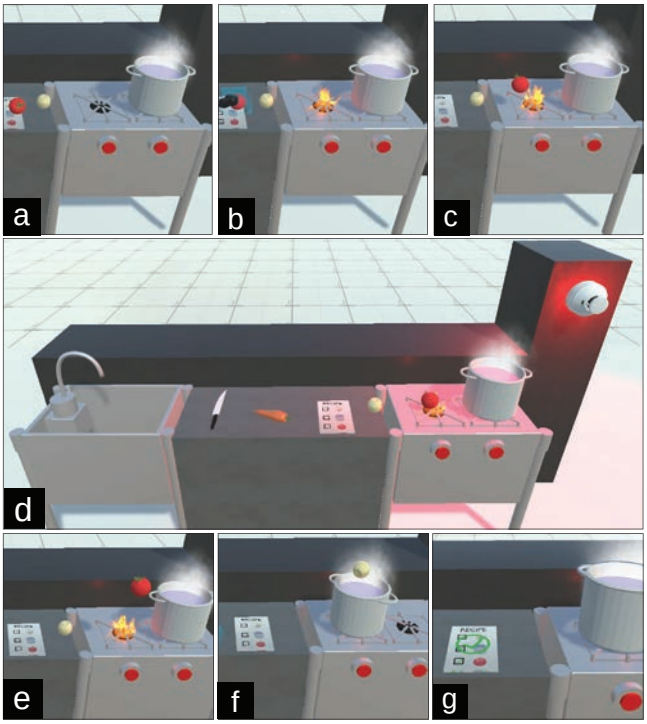


Figure 8: (*a*) At some point in Simulation mode. (*b*) The Tomato gets grabbed. (*c*) The Tomato is moved over to the Stove’s fire. (*d*) The Fire Alarm goes off. (*e*) The Tomato gets moved from the Stove’s fire to the Pot. (*f*) The Onion gets moved into the Pot. (*g*) Finally, the recipe gets marked as Done.

the Suggestions Gallery, confirming their current state or choosing a suggestion and adding those to the current program.

6.2 Case Study 2: Car and Crosswalk Simulator

Our second case study is inspired by works in Human-Robot Interaction literature [20] where simulation environments for predictability and model testing come in handy. A similar scene can also be used in a game or virtual experience. In Figure 9 we have programmed that when the orange Avatar walks into the influence of the Crosswalk, the Walking Sign changes from "Stop" to "Walk". And if the Sign changes to "Walk" and a Car is in influence of the Crosswalk, then it stops moving. This scenario showcases assets that have motion animations (moving characters and cars), and the ease AuthorIVE provides to author interactions in a type of "frozen" state while in demonstration mode, and then testing out the created programs in simulation mode. This also shows that the interactions created are entirely up to the author, and don't necessarily have to follow any game logic. For example, in Figure 9 the Cars have no awareness of the traffic signal and only stop moving if the orange Avatar reaches the Crosswalk and the Walking Signal changes to "Walk". The authored program could have prioritized the traffic instead and only had the Avatar cross the road if the traffic signal was red (instead of the other way around).

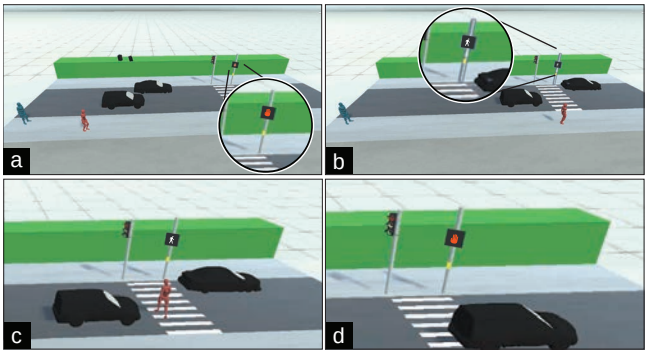


Figure 9: (*a*) Orange Avatar is walking and the Walk Sign is on "Stop". (*b*) Orange Avatar walks into the influence of the Crosswalk and the sign changes to "Walk", and the Cars stop. (*c*) Orange Avatar crosses the Crosswalk. (*d*) The Sign changes back to "Stop" and Cars resume moving.

6.3 Case Study 3: Balloon Shooting Game

The third case study emulates a very common 2D and VR game (such as in Valve's The Lab [28]). One of the Google Cardboard onboarding demonstrations walks the user through shooting at balloons using the gaze pointer. This type of game also inspired dozens of projects created on Scratch [22] that simulate it. In Figure 10 we demonstrate the creation of such game using AuthorIVE. The figure was created with one simple program:

```
Program:
    IN:<contact.Balloon.Dart>,
    OUT:<action.Balloon.Pop>
```

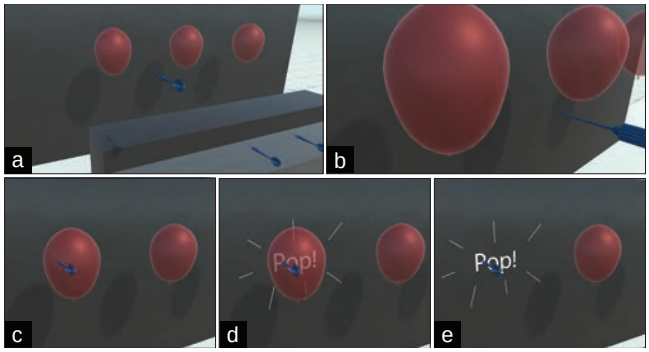


Figure 10: (*a*) The Dart is thrown towards the Balloon (*b*) A close up of the Dart entering the Balloon's influence zone. (*c*) The Dart has made contact with the Balloon. (*d,e*) The Balloon animates its Pop effect.

Here we experiment with authoring interactions that involve throwing objects and, once they intersect each other's contact bounds, executing the authored programs mid-air.

6.4 Case Study 4: Educational Trash Sorter

With the interest level of sustainability (as observed on Google Trends [6]) almost doubling in the past 5 years, we made our fourth case study an educational trash sorting game. As seen in Figure 11 if the object is placed in the incorrect Trash Bin, the bin glows red (Figure 11b), and when it is placed in the correct Trash Bin, the bin glows neon green (Figure 11d). For this scene each object has to be programmed individually, properly setting up each trash item with its corresponding Trash Bin. This case study exemplifies how a system like ours that allows simple interaction authoring can be used in a classroom setting or even to create more immersive educational experiences in times of social distancing.

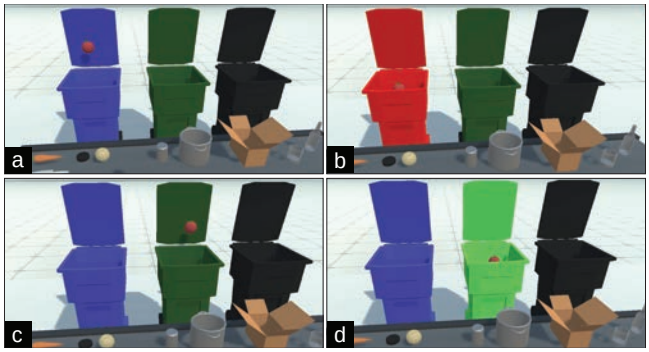


Figure 11: (*a*) The Tomato is brought over to the Recycling Bin. (*b*) As the Tomato gets dropped in, the bin glows red indicating: Wrong Selection. (*c*) The Tomato then gets moved over to the Compost Bin. (*d*) The Compost Bin glows green to indicate: Correct Selection.

7 LIMITATIONS

In spite of its flexibility, our system currently presents limitations that restrict the types of experiences that can be authored. The limitations are:

(1) We only handle three types of events: influence, contact and actions.
(2) All types of events are discrete and binary (i.e.: true or false).
(3) Program outputs are restricted to action events.

On the suggestions side, we are also partially limited despite the current heuristics, the Suggestions Gallery can feel too combinatorial at times. In order to handle this we propose a couple Future directions for this work.

8 FUTURE WORK

The decoupling between the World state, Gallery mechanism for showing suggestions and the Suggestions System to generate these suggestions makes it possible to address the system's current limitations and further extend it in a number of ways.

One interesting research direction we would like to pursue is reformulating the world in terms of Reinforcement Learning. This can make the system capable of handling arbitrary, continuous events in a statistical way, both for input and output states.

Particularly, we are excited about exploring Imitation and Apprenticeship learning to make fewer suggestions, consisting of complete programs, as opposed to multiple partial programs, which would alleviate cluttering in the Suggestions Gallery and open the door to one-shot demonstrations.

Another interesting extension of our work would be to use natural language embedding such as Word2Vec [17]. Using this would allow us to better resolve and suggest object actions based on the semantics of what they are called. For example, in the trash sorting case study maybe Tomato would be closer in distance to Compost than to Recycling resolving the need to individually program each item.

9 CONCLUSION

We present AuthorIVE, a VR interaction authoring tool allowing users to program experiences through disambiguating demonstrations. We showed that AuthorIVE is able to program fairly complete interactive experiences through a more direct interaction, that could reduce time and efforts required. Our case studies demonstrate the generative power and flexibility of AuthorIVE. While the initial usages of AuthorIVE are limited to pair-wise inter-object interaction programming, many current VR games and experiences can be recreated with it. We believe that our future directions provide a lot of potential to what a VR tool like this could become. An additional application of our tool could be as an extension to existing VR building tools (such as Unity), where it can be turned on and off when needed. This way it could not only guide a novice but also augment the current workflow of an experienced programmer.

The goal of AuthorIVE is to enable non technical creators to make virtual reality experiences that not only look amazing, but also have the interactions one might expect in an immersive experience. Building out these systems take time, skill, and patience, and hopefully with an authoring tool that is simple yet powerful enough, more interesting content will be created and made available. We hope that with a simple tool like AuthorIVE, more creatives and non-programmers can author amazing interactive experiences.

ACKNOWLEDGMENTS

Thanks to Brian Aronowitz for most of the 3D models and assets, Andrew Head for programming synthesis discussions, Jeremy Warner for algorithms and evaluation discussions, and Anna Brewer for help with initial brainstorming. We also used some simple assets from Unity's asset store, and other free online sources.

REFERENCES

[1] Rahul Arora, Rubaiat Habib Kazi, Danny M. Kaufman, Wilmot Li, and Karan Singh. 2019. MagicalHands: Mid-Air Hand Gestures for Animating in VR. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 463–477. https://doi.org/10.1145/3332165.3347942

[2] Olivier Bau and Wendy E Mackay. 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. 37–46.

[3] Blender3D [n.d.]. . https://www.blender.org/

[4] Allen Cypher and Daniel Conrad Halbert. 1993. *Watch what I do: programming by demonstration*. MIT press.

[5] Barrett Ens, Fraser Anderson, Tovi Grossman, Michelle Annett, Pourang Irani, and George Fitzmaurice. 2017. Ivy: Exploring Spatially Situated Visual Programming for Authoring and Understanding Intelligent Environments. In *Proceedings of the 43rd Graphics Interface Conference (GI '17)*. Canadian Human-Computer Communications Society, Waterloo, CAN, 156–162.

[6] Google Trends [n.d.]. *Sustainability Trend over past 5 years*. Retrieved May 25, 2020 from https://trends.google.com/trends/explore?date=today%205-y&geo=US&q=%2Fm%2F0hkst

[7] Sumit Gulwani. 2011. Automating String Processing in Spreadsheets Using Input-Output Examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '11)*. Association for Computing Machinery, New York, NY, USA, 317–330. https://doi.org/10.1145/1926385.1926423

[8] Brian Hempel and Ravi Chugh. 2016. Semi-automated svg programming via direct manipulation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 379–390.

[9] Caitlin Kelleher and Randy Pausch. 2007. Using Storytelling to Motivate Programming. *Commun. ACM* 50, 7 (July 2007), 58–64. https://doi.org/10.1145/1272516.1272540

[10] Gordon Kurtenbach and William Buxton. 1993. The Limits of Expert Performance Using Hierarchic Marking Menus. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. Association for Computing Machinery, New York, NY, USA, 482–487. https://doi.org/10.1145/169059.169426

[11] Tessa Lau, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. 2003. Programming by demonstration using version space algebra. *Machine Learning* 53, 1-2 (2003), 111–156.

[12] G. A. Lee, C. Nelles, M. Billinghurst, and G. J. Kim. 2004. Immersive authoring of tangible augmented reality applications. In *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*. 172–181.

[13] Gilly Leshed, Eben M Haber, Tara Matthews, and Tessa Lau. 2008. CoScripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1719–1728.

[14] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. 1997. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '97)*. ACM Press/Addison-Wesley Publishing Co., USA, 389–400. https://doi.org/10.1145/258734.258887

[15] Mikaël Mayer and Viktor Kuncak. 2013. Game programming by demonstration. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*. 75–90.

[16] Mikaël Mayer, Gustavo Soares, Maxim Grechkin, Vu Le, Mark Marron, Oleksandr Polozov, Rishabh Singh, Benjamin Zorn, and Sumit Gulwani. 2015. User interaction models for disambiguation in programming by example. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 291–301.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13)*. Curran Associates Inc., Red Hook, NY, USA, 3111–3119.

[18] Mindesk [n.d.]. . https://mindeskvr.com/
[19] Oculus Blog 2019. *Rift Creation Spotlight: Building Oculus Home with Custom Objects and Models.* Retrieved May 25, 2020 from https://www.oculus.com/blog/rift-creation-spotlight-building-oculus-home-with-custom-objects-and-models/
[20] Dorsa Sadigh, Shankar Sastry, Sanjit A. Seshia, and Anca D. Dragan. 2016. Planning for Autonomous Cars that Leverage Effects on Human Actions. In *Proceedings of Robotics: Science and Systems*. AnnArbor, Michigan. https://doi.org/10.15607/RSS.2016.XII.029
[21] Nazmus Saquib, Rubaiat Habib Kazi, Li-Yi Wei, and Wilmot Li. 2019. Interactive Body-Driven Graphics for Augmented Video Performance. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, Article Paper 622, 12 pages. https://doi.org/10.1145/3290605.3300852
[22] Scratch Projects [n.d.]. *Search Projects: Balloons.* Retrieved May 25, 2020 from https://scratch.mit.edu/search/projects?q=ballons
[23] Brenden Sewell. 2015. *Blueprints Visual Scripting for Unreal Engine.* Packt Publishing Ltd.
[24] David Canfield Smith, Allen Cypher, and Jim Spohrer. 1994. KidSim: programming agents without a programming language. *Commun. ACM* 37, 7 (1994), 54–67.
[25] David Canfield Smith, Allen Cypher, and Larry Tesler. 2000. Programming by Example: Novice Programming Comes of Age. *Commun. ACM* 43, 3 (March 2000), 75–81. https://doi.org/10.1145/330534.330544
[26] Unity 2019.3.7f1 [n.d.]. . https://unity.com/
[27] Unreal Engine [n.d.]. . https://www.unrealengine.com/en-US/
[28] Valve Software [n.d.]. *The Lab game.* Retrieved May 28, 2020 from https://www.valvesoftware.com/en/
[29] Kuat Yessenov, Shubham Tulsiani, Aditya Menon, Robert C. Miller, Sumit Gulwani, Butler Lampson, and Adam Kalai. 2013. A Colorful Approach to Text Processing by Example. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. Association for Computing Machinery, New York, NY, USA, 495–504. https://doi.org/10.1145/2501988.2502040
[30] R. Zollner, O. Rogalla, R. Dillmann, and M. Zollner. 2002. Understanding users intention: programming fine manipulation tasks by demonstration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2. 1114–1119 vol.2.