

Understanding Sketch-and-Speech Descriptions of Machines

by

Jeremy Scott

B.A.Sc. in Engineering Science, University of Toronto (2010)

Submitted to the Department of Electrical Engineering and Computer
Science

in partial fulfillment of the requirements for the degree of

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

© Jeremy Scott, MMXII. All rights reserved.

The author hereby grants to MIT permission to reproduce and to
distribute publicly paper and electronic copies of this thesis document
in whole or in part in any medium now known or hereafter created.

Author
Department of Electrical Engineering and Computer Science
August 31, 2012

Certified by
Randall Davis
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by
Professor Leslie A. Kolodziejcki
Chairman, Department Committee on Graduate Theses

Understanding Sketch-and-Speech Descriptions of Machines

by

Jeremy Scott

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2012, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

In this thesis, we present PhysInk, a sketching application that provides a natural way to describe physical structure and behavior to a computer. The user describes structure by sketching physical objects and constraints on a canvas backed by a 2D physics engine. The user can then move the objects to explicitly demonstrate physical behavior and can utter simple speech commands to label objects and events. These interactions are captured in PhysInk's timeline - a causal graph of physical events - which is used to build a deeper understanding of behavior. The events capture the geometry of movement and contact between objects, while the timeline captures the causal relationships between events. Capturing behavior in this framework enables a more intelligent conversation with the user about function. First, it enables assisted editing, where PhysInk propagates the user's edits of behavior through the timeline in order to maintain a logical sequence of events at all times. Second, it allows users to produce a physically-realistic simulation of the behavior that they have described. These features demonstrate the usefulness and depth that the timeline offers as a knowledge representation for behavior, making PhysInk the first in a new class of design tools that focuses on function, as well as form.

Thesis Supervisor: Randall Davis

Title: Professor of Computer Science and Engineering

Acknowledgments

I would like to thank my advisor, Professor Randall Davis, first for his insight, support and encouragement, second for always reminding me of the vision driving this thesis, and third for steering me clear of many bottomless pits.

I would like to thank the NSERC PGS program for their financial support.

Thank you also to Professor Maria Yang for an immensely helpful discussion on the utility of PhysInk in the product design world. Thank you to Tom Ouyang for his guidance during the early stages of this project. Thank you to Andrew Correa for sharing his implementation of the PaleoSketch system. Also, thank you to Chih-yu, Ying, Yale and Danica for their feedback and support.

Finally, thank you to my friends, my family, Jennifer and my parents for giving me their constant encouragement and confidence over the past two years.

Contents

1	Introduction	15
1.1	Motivation and Overview	15
1.2	Contributions	17
1.3	Thesis Outline	17
2	PhysInk Functionality	19
2.1	Sketches of Physical Structure and Behavior	19
2.2	Sketch Mode: Describing Structure	23
2.2.1	Sketching Bodies and Constraints	23
2.2.2	Adjusting Positions and Orientations	25
2.2.3	Labeling Structure	26
2.3	Demo Mode: Capturing Behavior	27
2.3.1	Demonstrating Behavior	27
2.3.2	Playing back, Saving and Sharing Behavior	30
2.3.3	Behavior as a Timeline	30
2.3.4	Narrating Events	31
2.3.5	Assisted Editing of Behavior	31
2.4	Finding Equivalent Behavior	33
2.4.1	Behavior Vocabulary	36
2.4.2	Physically-Realistic, Equivalent Behavior	37
3	Implementing PhysInk	39
3.1	Architecture Overview	39

3.2	Understanding Structure	40
3.2.1	Stroke Recognition and Beautification	41
3.2.2	Context-Sensitive Interpretation	42
3.2.3	Physical Instantiation	43
3.2.4	Adjusting Structure	45
3.2.5	Labeling Components	46
3.3	Understanding Behavior	47
3.3.1	The Timeline	48
3.3.2	Building the Timeline	52
3.3.3	Labeling Events	59
3.4	Finding Physically-Realistic Behavior	60
3.4.1	Review: Qualitative Equivalence	61
3.4.2	The Simulator	61
3.4.3	Seeding Parameters	63
3.4.4	Evaluating Equivalence	64
3.4.5	Adjusting Parameters	65
3.4.6	Examples	65
4	A Portfolio of Machines	67
4.1	Basketball Collision	67
4.2	Driving Range Golf Ball Helper	69
4.3	Cup Dispenser	71
5	Related Work	75
5.1	Systems for Capturing Structure	75
5.2	Animation-by-Demonstration	76
5.3	Systems for Capturing Behavior	77
6	Discussion	79
6.1	Future Work	79
6.1.1	Supporting More Types of Behavior	79

6.1.2	Three-Dimensional Behavior	80
6.1.3	Improving the Search for Physically-Correct Behavior	80
6.2	Applications	81
6.2.1	Product Design	81
6.2.2	Video Game Design	81
6.2.3	Programming and Other Domains	81
7	Conclusion	83

List of Figures

2-1	Overview sketch of the egg cracker's structure.	20
2-2	Storyboard illustrating the egg cracker's behavior.	20
2-3	The PhysInk UI	22
2-4	Sketching the egg cracker's parts	23
2-5	Sketching the egg cracker's constraints	24
2-6	Adjusting the egg cracker's structure	26
2-7	Demonstrating the egg cracker's behavior	27
2-8	Describing concurrent movement	29
2-9	PhysInk's button menu	29
2-10	Demonstrating the pinball machine's behavior	32
2-11	Changing the pinball machine's behavior	33
2-12	Examples of inequivalent behavior: Different Events	34
2-13	Examples of inequivalent behavior: Contact surfaces	35
2-14	Examples of inequivalent behavior: Relative motion	35
2-15	An example of equivalent behavior.	36
2-16	Demonstrating a simple behavior	37
2-17	User-described behavior and physically-realistic, qualitatively equivalent simulation	38
3-1	PhysInk Architecture	40
3-2	Processes for Capturing Structure	41
3-3	Processes involved in capturing behavior.	47
3-4	The egg cracker's timeline	48

3-5	Movement event data structure	49
3-6	Qualitative Geometry of Movement: Relative Motion	50
3-7	Contact event data structure	51
3-8	Qualitative Geometry of Contact: Surfaces in Contact	51
3-9	End-contact event example	52
3-10	Pinball machine's behavior	53
3-11	Building the pinball machine's timeline.	54
3-12	Updating the ball's trajectory.	55
3-13	The pinball machine's timeline.	56
3-14	Modified pinball machine	56
3-15	Editing Behavior: The original behavior	57
3-16	Editing Behavior: Changing the trajectory	58
3-17	Editing Behavior: The new overall behavior	58
3-18	Editing the pinball machine's behavior.	59
3-19	The simulator's main steps	62
3-20	Seed vector for ball's initial velocity.	63
3-21	Finding Physically-Realistic Behavior: Example 1	65
3-22	Finding Physically-Realistic Behavior: Example 2	66
4-1	Demonstrating Basketball and Concurrent Movement	68
4-2	Timeline for the basketball collision.	68
4-3	The user's description and the simulation of the basketball collision	69
4-4	The golf ball helper's structure.	70
4-5	The description and simulation of the golf ball helper's behavior	70
4-6	The cup dispenser's structure.	71
4-7	Adjusting the cup dispenser's structure	72
4-8	Describing the cup dispenser's behavior	73
4-9	Physically-realistic, qualitatively equivalent simulation of Cup Dispenser	73

List of Tables

2.1	Components and constraints that can be sketched in PhysInk.	25
-----	---	----

Chapter 1

Introduction

1.1 Motivation and Overview

Capturing and communicating behavior is an important task in a variety of domains. In product design for example, engineers discuss a product's behavior in terms of the different physical mechanisms that lead to its functionality. During brainstorming, these mechanisms are sketched on pen-and-paper and whiteboards. A team of designers can quickly modify the sketches, discuss ideas and make design decisions.

Unfortunately, these sketches are static and depicting movement requires the use of arrows or storyboards, leading to ambiguity and time wasted on redrawing the device in successive frames. Although animation is the obvious next step for portraying physical behavior, general animation software (Flash, K-Sketch [9]) lacks physicality. For example, if a mechanism involves a ball rolling down a ramp, the animation software has no notion of contact and the user must ensure that the ball and ramp do not overlap at each frame. Animating physical behavior can therefore be an arduous task if the user wants to maintain any sense of realism.

At the other end of the spectrum, traditional CAD tools support physicality, but focus on structure rather than behavior. The user must specify form that implies function, rather than function that implies form. These tools are useful for modeling the device after exact dimensions, angles and fixtures between its parts have been worked out. The result is a 3D model that gives a high sense of realism, but forces

the designer to have a mature concept in place beforehand. Although it is possible to produce animations using the model, the leap from informal sketching to 3D-modeling is jarring when the designer wants to discuss behavior instead of structural details. Furthermore, traditional CAD tools generally have a high learning curve compared to sketching where the designer can freely and abstractly express his or her ideas on a piece of paper.

In this thesis, we present PhysInk, a sketching application that provides a natural way to describe both physical structure and behavior to a computer. PhysInk feels almost as natural as pen-and-paper, because the user can roughly sketch a device's structure on a canvas. Unlike real paper, however, PhysInk interprets the user's strokes as physical objects and constraints in a 2D world. The constraints enforce physically-realistic limitations on how an object can move - for example, an object with a rotational joint can rotate but cannot translate.

The user can then demonstrate the device's behavior by directly moving its parts on the canvas, knowing that they will move only in physically-expected ways. This produces the physicality lacking in animation software and avoids the structural details required by traditional CAD tools.

As the user describes behavior, PhysInk captures the motion and contact of objects in a timeline of physical events. As a result, PhysInk is not only recording frames of graphics (as done in animation), but is also capturing the events and causality involved in the behavior. Capturing behavior in timelines enables PhysInk to engage in a more intelligent conversation with the user about function. First, it enables assisted editing, where PhysInk propagates changes in behavior through the timeline in order to maintain a logical sequence of events at all times. Second, it allows users to produce physically-realistic simulations of behavior that they have demonstrated a single time.

The captured description can be labeled and narrated using simple speech commands, and can be exported for archiving and sharing. We see PhysInk as an intuitive user interface for capturing behavior, and the first in a new class of software that goes beyond solely recording graphics or physical structure.

1.2 Contributions

This thesis makes three contributions towards the fields of human-computer interaction and intelligent user interfaces:

1. We build the first CAD-like tool that focuses on soliciting and capturing descriptions of physical behavior rather than structure. We introduce the timeline, a data structure for capturing physical behavior in terms of physical events and causality between events. This in turn allows the system to build a deeper understanding of behavior and engage in a more intelligent conversation with the user about function as well as form.
2. We demonstrate how well the system captures the user-described behavior by using the timeline to provide feedback to the user. This feedback is given through two features: (1) assisted editing, where PhysInk adjusts the captured description in response to the user's changes in order to ensure it is causally sound, and (2) finding physically-correct simulations of the user-described behavior. The degree to which the simulation 'matches' the originally described behavior is a measure of how well PhysInk has understood the user's description. The implementation of these features demonstrates both the fidelity and utility of timelines.
3. Lastly, we combine a system for recognizing geometric primitives with a physics engine, in order to provide an intuitive interface for describing physical movement. This demonstrates how AI can be integrated into a user interface to preserve the feel of pen-and-paper, while offering digital archiving and intelligent UI features like physicality.

1.3 Thesis Outline

In the next chapter, we present PhysInk's user interface through its two modes of operation, sketch mode and demo mode, which can be used to describe a device's

structure and behavior, respectively. We also discuss a vocabulary for describing behavior that can be used to decide whether two behaviors are equivalent or inequivalent. In chapter 3, we discuss the implementation of PhysInk and our approach to recognizing strokes, instantiating physical objects on the canvas, and capturing behavior in timelines. We also discuss how PhysInk is able to demonstrate its understanding by assisting the user's edits and finding physically-realistic versions of the behavior that they have described. In chapter 4, we present a portfolio of machines that can be described using PhysInk. Finally, we overview related work in chapter 5, discuss future work in chapter 6 and summarize the thesis in chapter 7.

Chapter 2

PhysInk Functionality

In this chapter, we first review the challenges involved in handling sketch-and-speech descriptions of physical behavior. We then present the PhysInk user interface and its two modes: sketch mode for describing the device’s structure and demo mode for recording the device’s behavior. Throughout, we explain the user interface through examples of Rube Goldberg-style machines, which accomplish simple tasks in whimsical, convoluted ways. Lastly, we discuss what it means to ‘understand’ behavior and PhysInk’s ability to produce a physically-realistic version of what the user has described.

2.1 Sketches of Physical Structure and Behavior

In product design, simple 2D sketches can be used to explore the various ways a device’s functionality can be realized. Generally, designers seek to convey a high-level vision of the behavior involved in a physical mechanism, and discuss the device’s structure in only as much detail as is necessary. Consider the sketches in Figures 2-1 and 2-2, where a product designer has illustrated the structure and behavior of an egg cracker.

The egg cracker is an example of a Rube Goldberg machine, where a simple task is accomplished in a whimsical fashion, through a series of simple physical interactions. The designer has described the egg cracker through an overview diagram, where

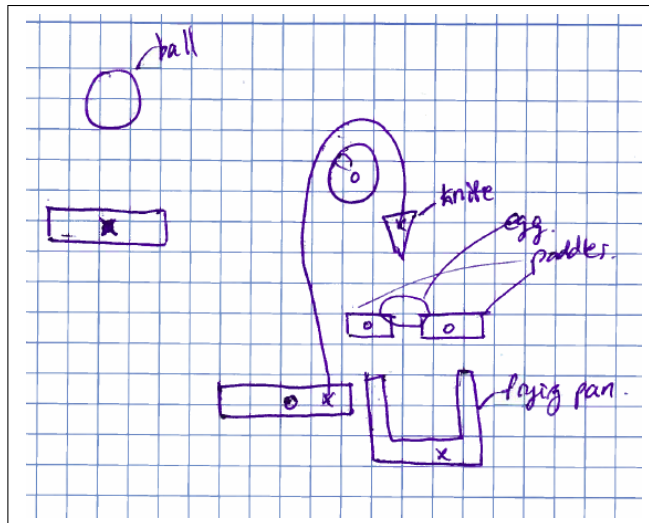


Figure 2-1: Overview sketch of the egg cracker's structure.

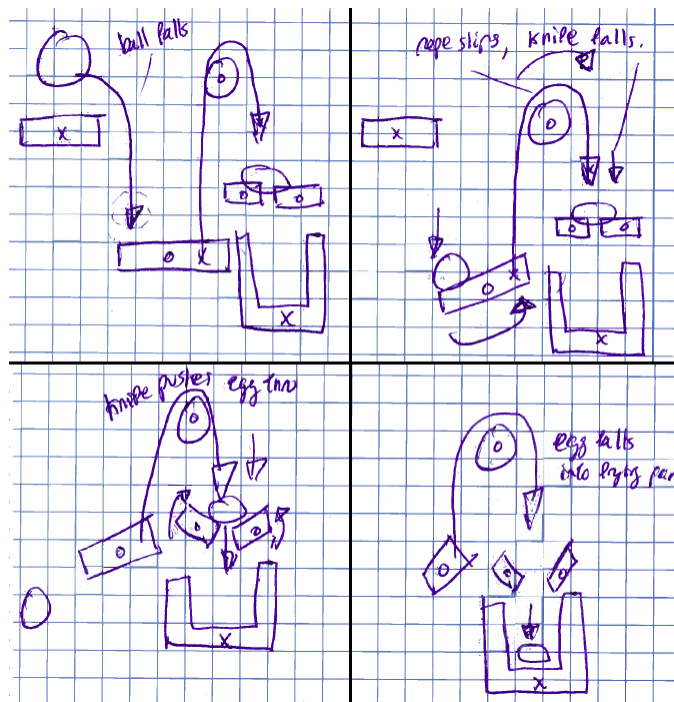


Figure 2-2: Storyboard illustrating the egg cracker's behavior.

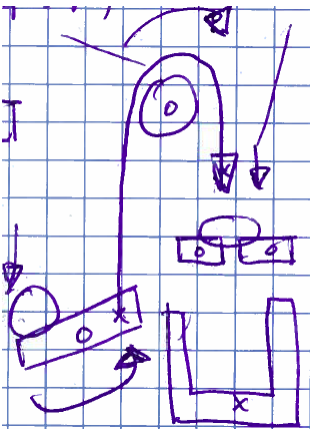
its rough structure is labeled, and a storyboard, where successive frames convey important events in the egg cracking mechanism. The designer explains that the ball lands on the rotating platform, causing it to rotate. The rotation causes the rope to rotate around the pulley, and the knife to drop onto the egg. This results in the egg being pushed through the paddles and into the frying pan.

The overview sketch illustrates structure roughly as the initial configuration of parts essential to describe the egg cracker’s behavior. It does not include exact angles or dimensions, and is vague about interconnections between the device’s parts. We can see that closed shapes generally represent rigid bodies (e.g. the platforms, knife, egg and ball). The unclosed curve connecting the platform to the knife represents a rope that loops around the pulley. Some bodies are attached to each other or to the background through various constraints. For example, X-symbols indicate that the first platform is fixed to the background (no translation or rotation) and a small circle in the rotating platform represents a rotational joint (no translation). In general, the level of detail gets no more advanced than this - threaded screws, brackets and ball bearings are abstracted into these high-level symbols.

Another important note with respect to structure is that sketching a physically-realistic or ‘correct’ representation of any device can be difficult. Here the designer wants to indicate that, initially, the egg is resting on the paddles. Drawing the egg and paddles so they are flush against each other is near impossible and taken literally, this sketch might imply that they are overlapping each other. While this ambiguity is handled easily by humans,



it can serve as a point of confusion for software systems that interpret sketches as physical structure. It is therefore important that these systems help users to clarify structural details.



In order to describe behavior the designer has laboriously drawn up a storyboard to convey a sequence of events. While arrows are sometimes used to represent movement, they can also be used to indicate possible degrees of freedom or callouts to explanatory handwriting. Furthermore, arrows can clutter the sketch after many rounds of discussion, leading to a convoluted diagram. These sources of ambiguity mean that designers must often redraw the device, which can be a time-consuming,

tedious process on pen-and-paper.

Ambiguity can be explained away during discussions, meaning that speech is essential to descriptions of physical behavior. For archiving and sharing purposes, product sketches are almost never without text labels for the device's parts. It is also important that descriptions of behavior (arrows and storyboards) are labeled with textual descriptions of events, particularly when these sketches are shared with co-workers absent from discussions.

PhysInk was built to assist with this task, allowing natural descriptions of behavior and rough diagrams of structure. It supports common sketching conventions for structure, interpreting polygons as rigid bodies, and X-symbols or circles as constraints on how the bodies can move. A physics engine was integrated into the interface, allowing movement and physical interactions to be demonstrated directly on the canvas. The physics engine also allows users to adjust the positions and orientations of objects to clarify structure. Finally, simple speech is recognized to allow labeling of objects and events.

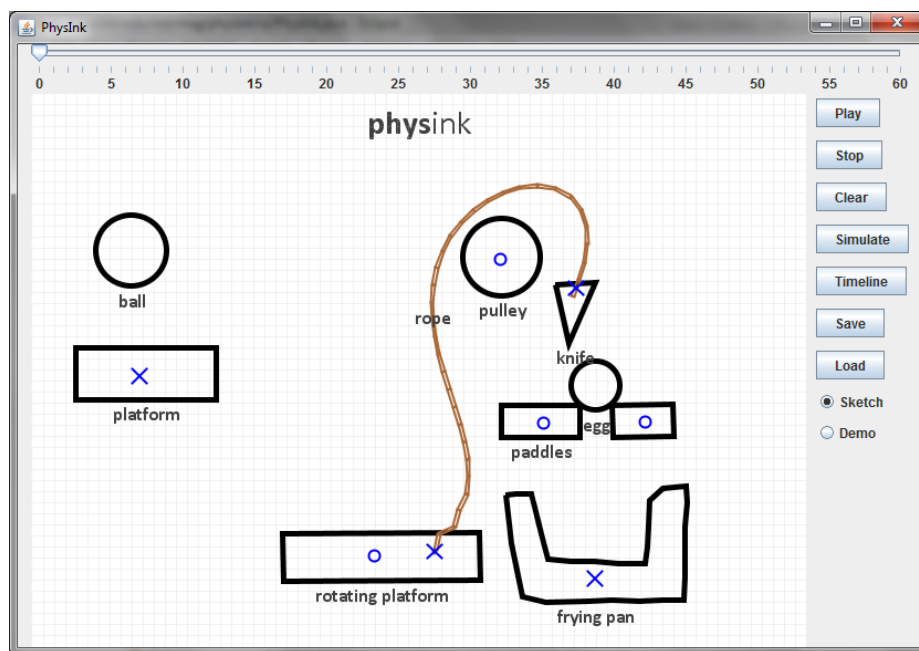


Figure 2-3: First look at PhysInk, a sketching application for describing physical structure and behavior.

2.2 Sketch Mode: Describing Structure

In sketch mode, the user can describe structure by drawing physical bodies and constraints on how they can move. The bodies' positions and orientations can be adjusted to resolve overlaps and contact. Simple speech and text entry can be used to label the diagram.

2.2.1 Sketching Bodies and Constraints

The user describes structure by sketching as they would on pen-and-paper. For example, in Figure 2-4 the user sketches the egg cracker's parts. Single stroke polygons and ellipses are interpreted as rigid bodies and beautified. The egg cracker's rotating platform is sketched roughly as a rectangle, but is interpreted as a rigid rectangular body and re-drawn cleanly.

The rope is created by drawing an unclosed stroke from the rotating platform to

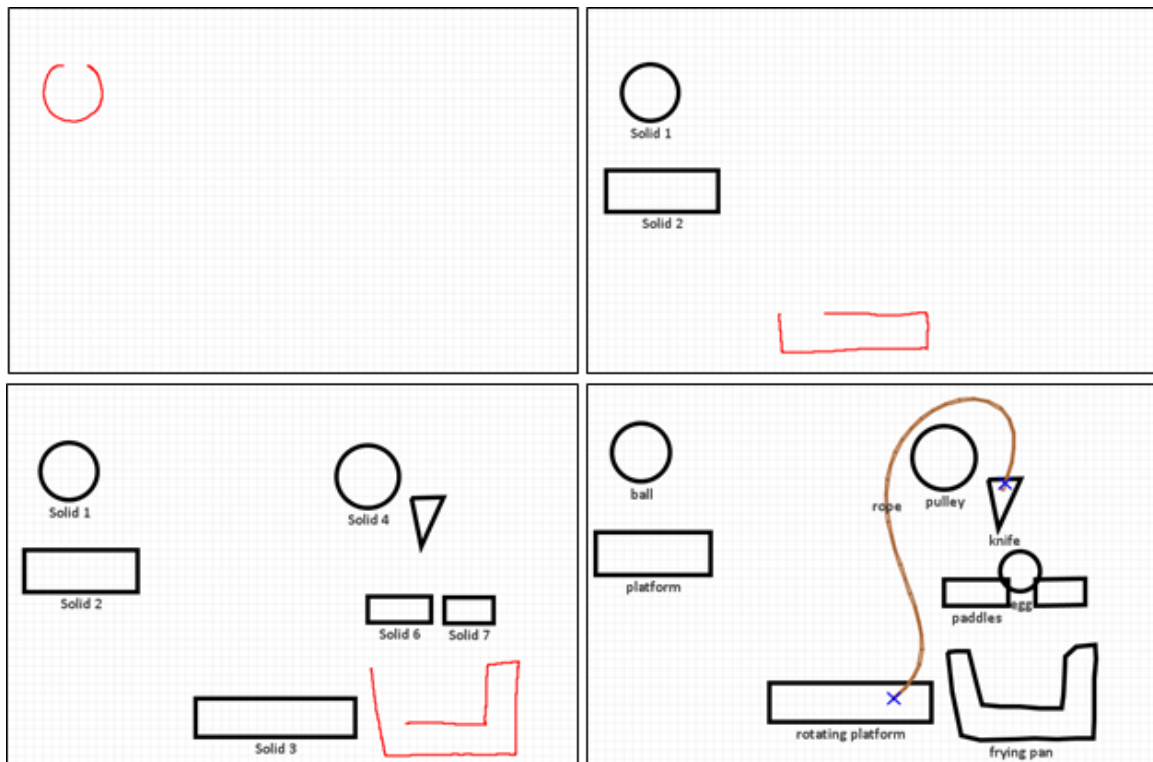


Figure 2-4: The user sketches the egg cracker's parts, as PhysInk beautifies and instantiates them in a 2D physics world.

the knife. Since the rope’s endpoints overlap with the platform and knife, PhysInk automatically fixes the rope to each object, saving some of the user’s time.

The user also indicates constraints on how the egg cracker’s parts can move by creating fixed and rotational joints (see Figure 2-5). For example, the first platform and the frying pan are fixed to the background using X-symbols. In addition to fixing objects to the background, X-symbols can be used to fix one body to another. PhysInk then ensures that the topmost object cannot translate or rotate with respect to the anchoring object. Similarly, rotational joints are created on the rotating platform, paddles and pulley by drawing small circles inside each object. PhysInk allows these objects to rotate, but prevents translation with respect to the anchoring object (the background, in this case).

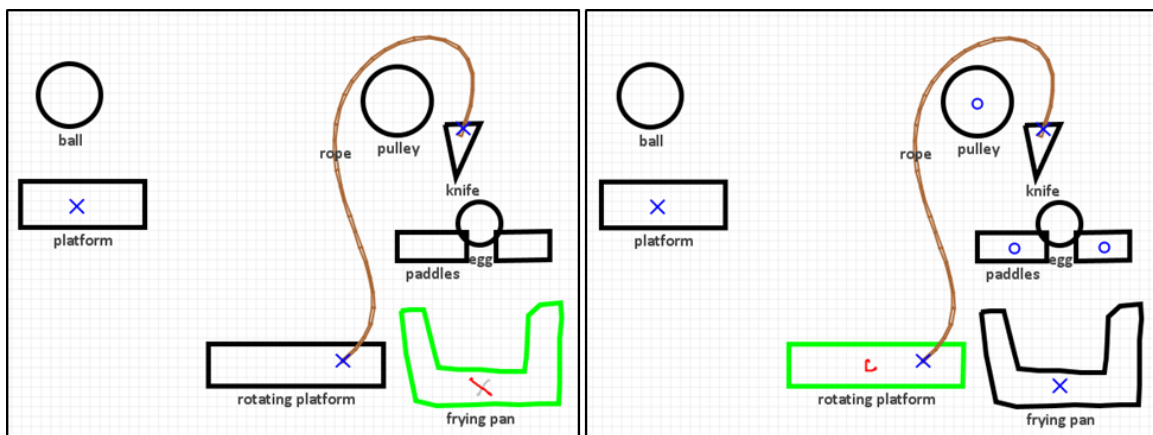


Figure 2-5: The user sketches constraints on the egg cracker’s parts.

The physical components and constraints that can be sketched in PhysInk are summarized in Table 2.1. Note that springs can be created by sketching helixes that start from an existing fixed body. In order to simplify the motion associated with springs, they are constrained to be perfectly horizontal or vertical, and can vibrate only along one axis. Also, by drawing a single stroke through an object, the user can erase a body from the canvas and at any time can undo (CTRL+Z) and redo (CTRL+R) their sketching interactions.

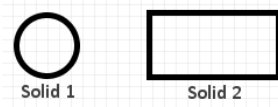
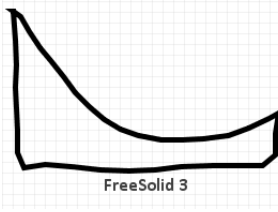
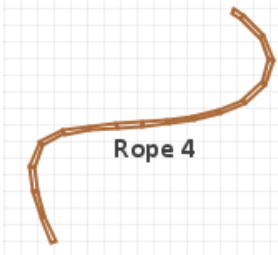
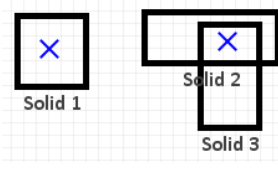
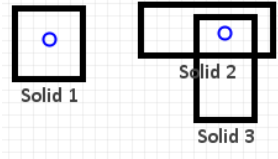
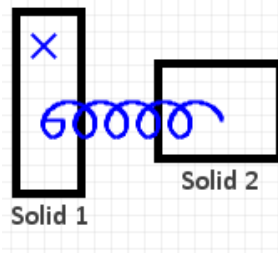
Entity	Sketch Input	PhysInk Interpretation
Rigid body	Elliptical, circular, or rectangular stroke	
Freeform rigid body	Any other closed stroke	
Rope	Unclosed stroke	
Fixed Joint	X-symbol inside existing body	
Rotational Joint	Small circle inside existing body	
Spring	Helix connected to at least one background-fixed body	

Table 2.1: Components and constraints that can be sketched in PhysInk.

2.2.2 Adjusting Positions and Orientations

The user can adjust the initial position and orientation of any object by holding the pen down inside its outline and dwelling until it becomes dark-green in color. They

may then move the object freely on the canvas, translating and rotating as though they had actually grabbed it at the selection point. If the point is closer to the center of mass, the user can translate the object without rotation, while selecting an edge is makes it easier to rotate without translating. Figure 2-6 illustrates the user resolving the overlap between the egg and the paddles.

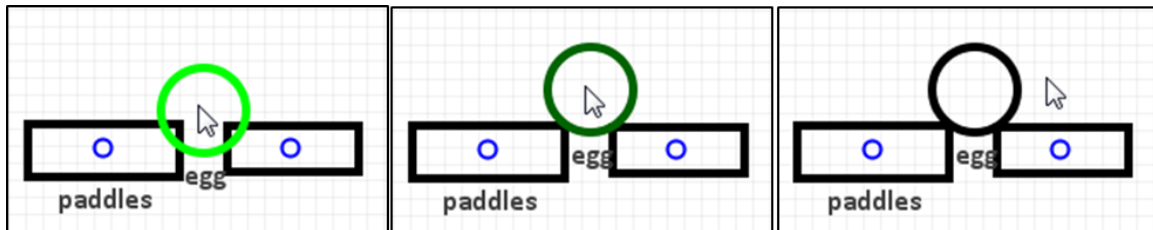


Figure 2-6: Adjusting the egg cracker's structure by first selecting and dwelling on the egg (until it turns dark green), and then moving it to resolve overlaps with the paddles.

Objects can be adjusted freely regardless of their physical constraints. For example, the rotating platform can be translated freely in sketch mode, even though it is anchored to the background. This allows designers to adjust positions and orientations, even after constraints have been applied to the bodies. When a particular object is being moved, all other objects are fixed and cannot be moved indirectly through contact. This allows the user to clarify which objects are in contact with each other and resolve overlaps, without affecting other objects' positions.

2.2.3 Labeling Structure

The user can label the egg cracker's parts by tapping an object with the pen and filling out a text entry box. Alternatively, simple speech is recognized and applied to the diagram. For example, while the user is sketching the ball they can state out loud: `label ball`. Speech can also be applied after sketching is complete by dwelling on an object and stating its label: `label platform`. These labels are displayed below the objects as they move on the canvas.

2.3 Demo Mode: Capturing Behavior

The user can then switch to demo mode, where sketched objects are moved on the canvas in order to demonstrate the device's behavior. Simple speech can be used to label events in the demonstration, which can be played back, saved and shared as an animation. PhysInk attempts to build a human-like understanding of the behavior by capturing it in a timeline of movement and contact events. The causality captured by the timeline leads to useful features, including assisted editing, where the user's changes are propagated through the timeline to maintain a logical ordering of events. The timeline also enables the user to generate a physically-realistic version of the behavior, which will be described in the next section.

2.3.1 Demonstrating Behavior

In demo mode, sketched objects are subject to the constraints described while in sketch mode. This produces an intuitive interface for describing behavior: the user

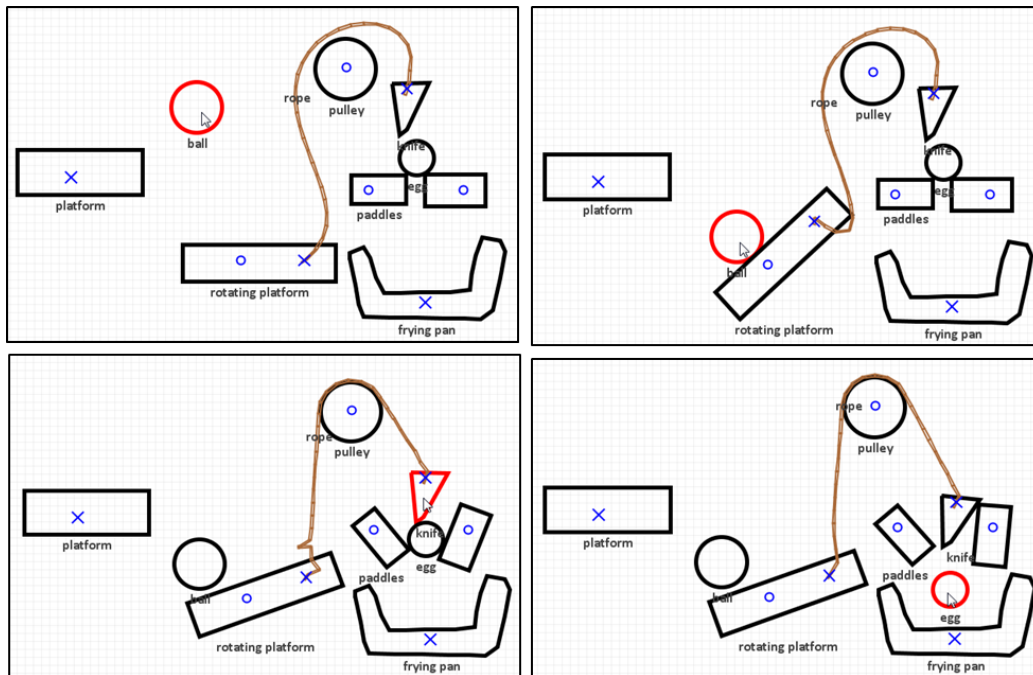


Figure 2-7: Demonstrating the egg cracker's behavior. The ball makes contact with the rotating platform (top row). The knife is then pulled down to show the egg being pushed into the frying pan (bottom row).

simply moves the objects in the way they are intended to behave. This interface contrasts starkly with other CAD tools, where the user incrementally describes structure that implies some desired behavior.

As shown in Figure 2-7, the user describes the egg cracker's behavior by explicitly moving the ball on a trajectory towards the rotating platform. As the ball and platform make contact, the platform rotates and the rope rotates around the pulley. Note that the user does not have to resolve the overlaps between rigid bodies as they would in traditional animation tools.

The user continues the description by pulling the knife down, making contact with the egg. Finally, the egg is moved through the rotating paddles and into the frying pan. All of the egg cracker's parts respond to force as they are meant to: fixed platforms are static and rotating objects are only allowed to rotate.

Zero Gravity

If PhysInk were only a physics simulator, the user would sketch the egg cracker's structure, press 'GO' and wait to see if gravity resulted in the egg falling into the frying pan. If not, they would have to keep adjusting the machine's structure until it yielded the desired behavior. Even in the simple case of the egg cracker, this becomes an arduous, repetitive process considering the user could be explicitly demonstrating the motion of the ball, platform, knife and egg.

Therefore, we decided that in demo mode there should be no gravity applied to the canvas. Other elements of physics are still in place, including contact between objects, tension in ropes, elasticity in springs and degrees of freedom implied by joints. This allows the user to freely describe motion trajectories, which may not be physically correct, but convey the machine's behavior effectively.

Concurrent Movement

Although only one object can be manipulated at a time, the user can rewind the animation and record the concurrent movement of other objects. For example, imagine the user wants to demonstrate the collision of two balls in mid-air as in Figure 2-8.

After describing the first ball's trajectory, the user can rewind to the beginning of the animation. When the user starts to move the second ball, the first ball's trajectory is played back concurrently, making it easy to describe the collision.

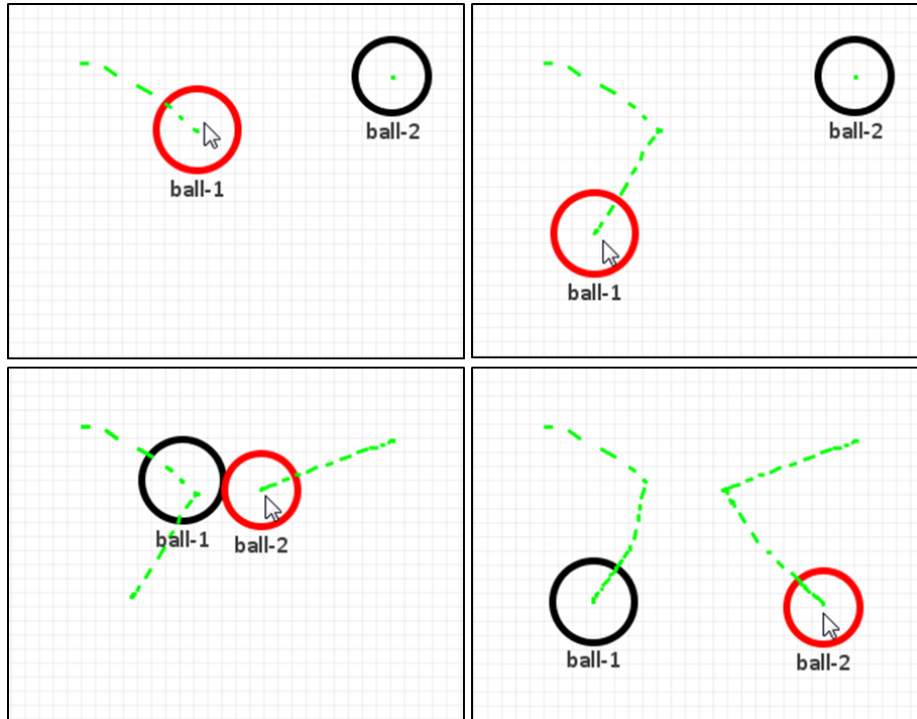


Figure 2-8: Describing concurrent movement. The trajectory of the ball 1 is demonstrated (top row). The user rewinds the animation, and then begins to describe the trajectory of ball 2 (bottom row). As it is moved, ball 1's trajectory is played back concurrently, so the collision can be easily described.

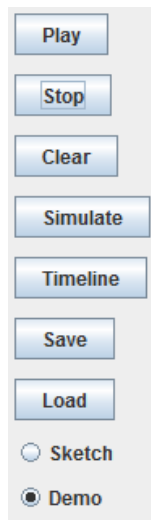


Figure 2-9: PhysInk's Button Menu. The PLAY/PAUSE and STOP buttons are used to control playback of the user's demonstration, while the CLEAR button erases the canvas. The SAVE and LOAD buttons allow the user to export and import previously described behaviors. The TIMELINE button (explained later) produces a textual description of the behavior as it is captured in the timeline. The SIMULATE button (explained later) allows the user to generate a physically-realistic version of the described behavior.

2.3.2 Playing back, Saving and Sharing Behavior

At any time, the user can play back their demonstration, using the PLAY/PAUSE and STOP menu buttons. The current frame of the animation is indicated by a slider bar, which can be used to rewind and fast-forward to specific points in time.

The SAVE button exports the behavior as an animation video, which can be shared and played back using any typical media player. PhysInk project files can also be saved and loaded (LOAD button), so that other users can edit the described mechanism and provide feedback.

2.3.3 Behavior as a Timeline

In order to build a deeper understanding of the user's description, behavior is captured in a timeline of movement and contact events between objects. At any time, the user can examine the timeline captured by PhysInk by pushing the TIMELINE button. This generates a textual description of the causal relationships between movement and contact events. For example, the egg cracker's timeline would appear as follows:

```
Movement of ball CAUSES
-- Contact between ball and rotating platform CAUSES
---- Movement of ball
---- Movement of rotating platform [attached to rope, knife] CAUSES
----- Contact between knife and egg CAUSES
----- Movement of knife
----- Movement of egg CAUSES
----- Contact between egg and paddles CAUSES
----- Movement of paddles
----- Movement of egg CAUSES
----- Contact between egg and frying pan
```

2.3.4 Narrating Events

As the behavior is played back, events are narrated along the bottom of the canvas, using the same labels included in the TIMELINE button's output. For example, as the ball is moving toward the rotating platform, an event label appears: "Movement of the ball."

Since these labels might be seen as lacking in descriptive power, the user can utter simple phrases to supply their own labels for events. This can be done while the event is being demonstrated, or afterwards by pausing the animation near the event that the user wants to label. The user gives a simple description for the event, involving two object names that were previously labeled in sketch mode:

```
<object> <verb> [<preposition>] [<object>]
```

For example, the user can speak as they move the egg cracker's ball: "the ball falls." As the egg makes contact with the frying pan the user can state: "the egg lands in the frying pan." PhysInk will then replace the appropriate event's subtitle with the user's label. Since PhysInk's initial vocabulary is limited, users can provide a supplementary text file with a list of verbs and prepositions that should be recognized.

2.3.5 Assisted Editing of Behavior

PhysInk can assist the user in editing their description of behavior. Consider the task of describing a pinball machine, where a ball hits a series of rotating paddles (Figure 2-10). After sketching its structure in sketch mode, the user switches to demo mode and moves the ball to demonstrate collisions with paddles A, B and C. Based on these interactions, PhysInk builds a timeline of movement and contact events between the ball and paddles.

After playing back the timeline, the user might decide that the behavior should be changed. Specifically, editing the description so that the ball hits paddle B on the right side of its pivot, causing it to bounce away without hitting paddle C (Figure 2-11). The user can edit the animation by first using the slider bar to rewind to the

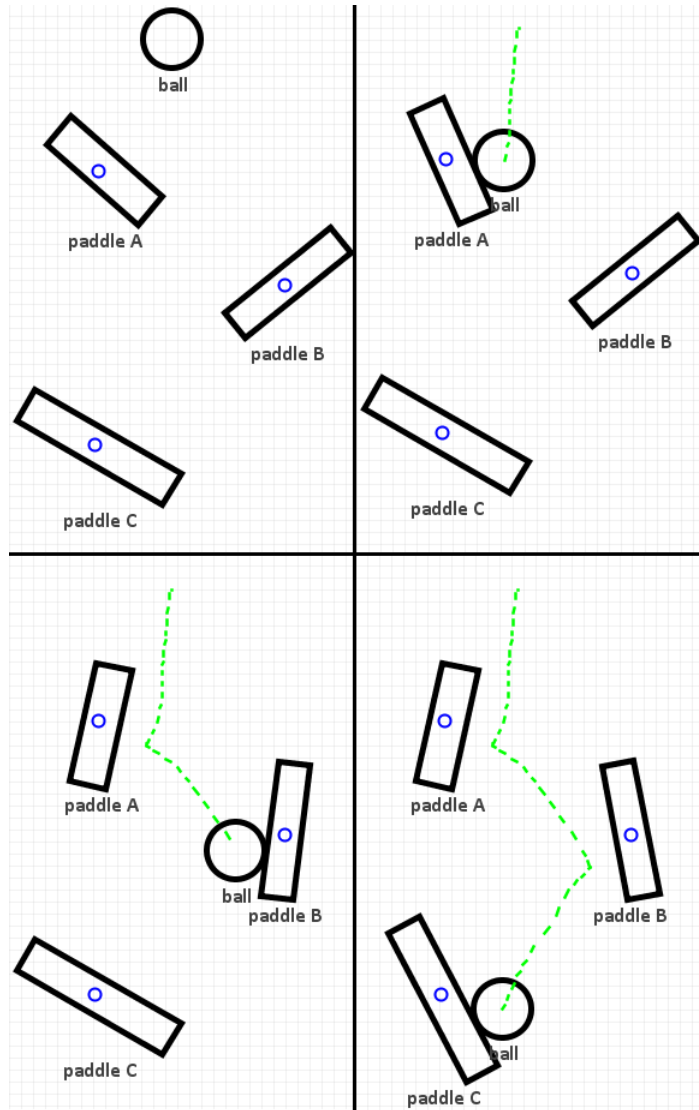


Figure 2-10: The pinball machine’s physical behavior, as demonstrated by the user.

point after the ball hits paddle A. At this point, the user changes the ball’s trajectory so it hits paddle B and bounces away to the right.

In a traditional animation tool, the movement of the ball and paddles would be recorded frame-by-frame, without any understanding that paddle C’s movement is caused by the ball making contact with it. As a result, even if the ball was deleted from the animation, the platform would still rotate, seemingly pushed by some invisible force.

In PhysInk, however, the timeline captures *causality* in the user’s demonstration. This allows the system to reason that if the ball’s movement is changed mid-trajectory,

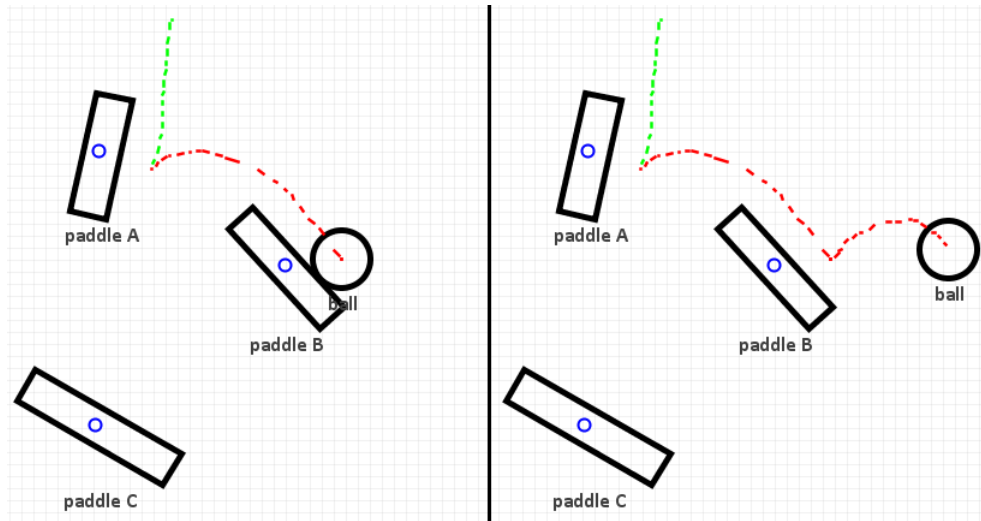


Figure 2-11: Changing the ball's behavior in the pinball machine.

then paddle C should not rotate because the ball no longer strikes it. Note in Figure 2-11 that paddle C remains stationary as the ball moves off to the right.

In other words, PhysInk can reason that events preceding a change are unaffected, but the events following a change may no longer be logical and should therefore be deleted. It can also reason that unrelated events are unaffected by changes and should not be deleted. For example, if a second ball were falling at the same time, adjacent to the paddles, its movement would remain untouched by this edit.

This ability to assist the user in editing behavior saves them time and effort, where they would otherwise need to erase and re-record the entire animation. The result is a seamless, intuitive interface for not only recording motion, but describing and editing behavior.

2.4 Finding Equivalent Behavior

We have explained that the timeline captures causality and how this knowledge can help to create an intuitive interface for the user. Still, one of PhysInk's main goals is to build behavioral knowledge that reflects the user's understanding. An important aspect of understanding behavior is knowing when one behavior is similar to another, or conversely, when two behaviors are inequivalent. Consider the simple mechanism

in Figure 2-12, where a ball can take a number of different paths before landing in a basket.

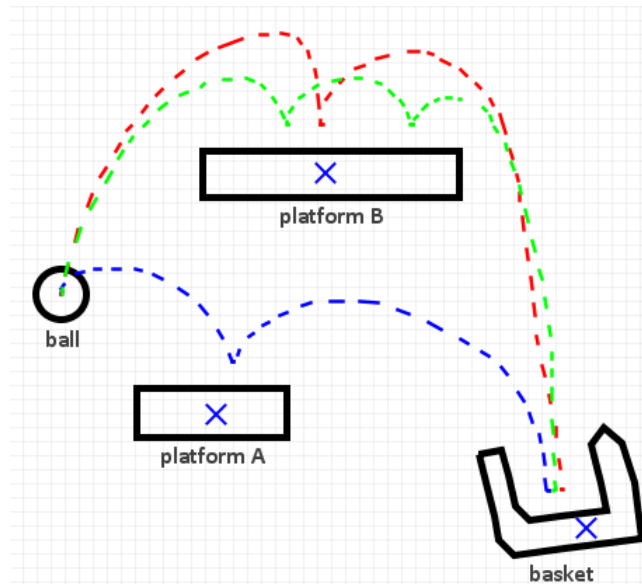


Figure 2-12: Behaviors may be considered inequivalent if the events and the ordering of events are not the same.

To the user, it is clear that these behaviors differ in the way that the ball is placed into the basket. We see the blue behavior as different from the red and green behaviors, because the ball hits platform A instead of platform B. Comparing just the red and green behaviors, we consider them inequivalent because the ball bounces once instead of twice, even though it only hits platform B in both cases. Clearly, for two behaviors to be equivalent, the events and the order of events (causality) must match. For two events to be equivalent, they must involve the same objects - yet this is not the whole story.

Consider the behaviors illustrated in Figure 2-13. In both of these behaviors the ball bounces off platform B once before landing in the basket. However, we still consider these behaviors to be inequivalent, because the ball makes contact with the top (red) instead of the bottom (blue) of the platform. Therefore, equivalence between contact events also concerns which sides of objects are involved.

Finally, consider the two paths in Figure 2-14, where the ball lands directly in the basket without touching the platforms. The behaviors are once again inequivalent

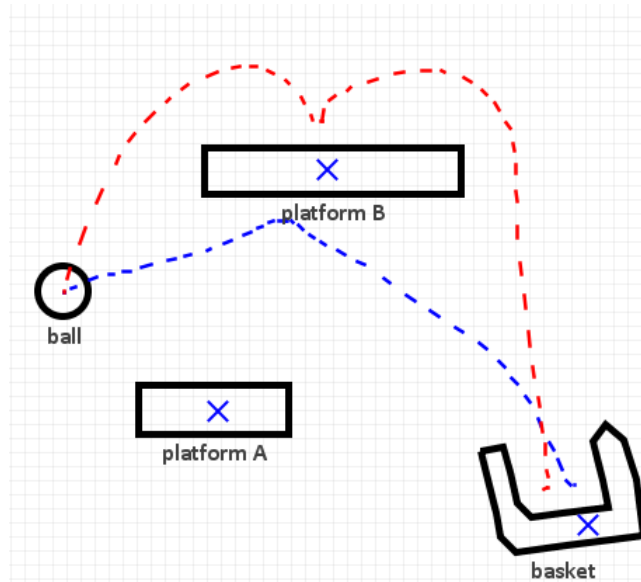


Figure 2-13: The side of an object involved in contact events is also important for determining equivalence.

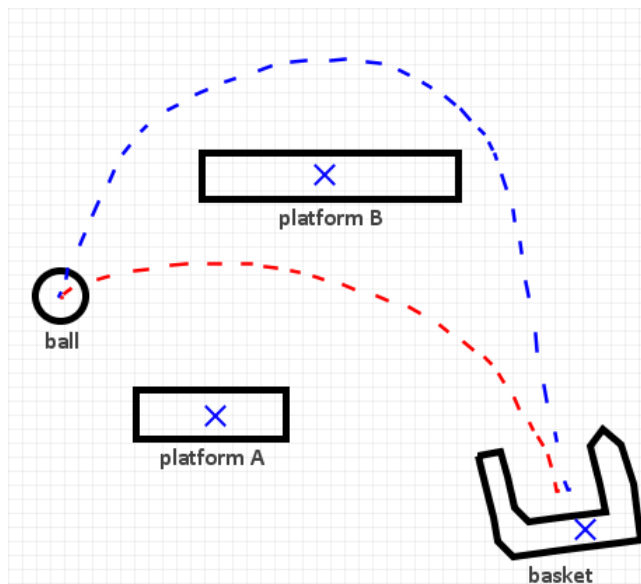


Figure 2-14: The motion of objects relative to each other (above, below) is also considered a factor in whether behaviors are inequivalent.

because in one case the ball passes above platform B, while in the other it passes below platform B. Therefore, equivalence between movement events is determined by the motion of an object relative to other objects in the scene.

In Figure 2-15, we illustrate an example of equivalent behavior. First, the sequence of events is the same between the two behaviors: the ball hits platform A, then

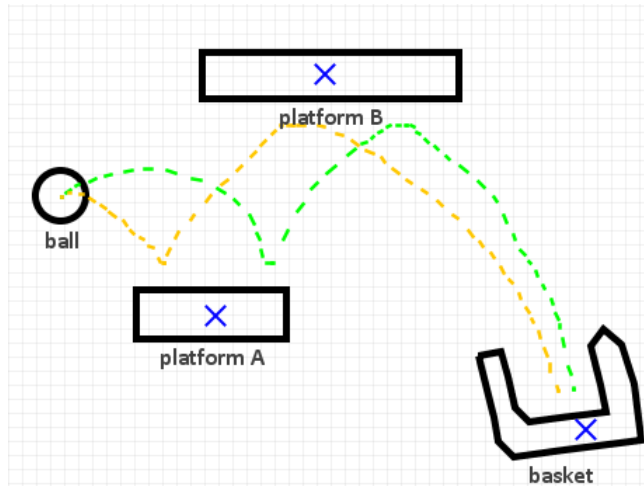


Figure 2-15: An example of equivalent behavior.

rebounds off platform B before landing in the basket. In both cases, the ball's motion relative to the platforms is the same. Also, in both cases the ball bounces off the top of platform A and the bottom of platform B. Although the trajectories are not exactly identical, to a human observer the two behaviors would be considered equivalent.

2.4.1 Behavior Vocabulary

These observations form a 'vocabulary' for describing one instance of behavior, and evaluating whether or not it is equivalent to another. First, a description consists of a set of movement and contact events that are linked together by *causality*. The events and the ordering of events can be compared between two descriptions of behavior to check for equivalence.

The events themselves are not described in terms of exact coordinates, but instead by what we term their *qualitative geometry*. For example, contact events are described by noting the sides of objects that hit each other, rather than focusing on exact collision locations. Movement events are also not described using exact trajectories, but instead in terms of an object's relative motion: does it pass above, below, left or right of nearby objects?

This vocabulary can be used by software systems to represent physical behavior. The system's understanding can be evaluated by asking it if two behaviors are

equivalent, and comparing the result to the user's expectations. Choosing the right vocabulary is important: if the choice is poor, then the system will think two behaviors are equivalent or inequivalent, where the user might disagree.

2.4.2 Physically-Realistic, Equivalent Behavior

PhysInk uses this vocabulary to capture the user's interactions in a causally-linked timeline of events. The system demonstrates how well it has captured the user's description by finding an equivalent version of the behavior that is physically-realistic.

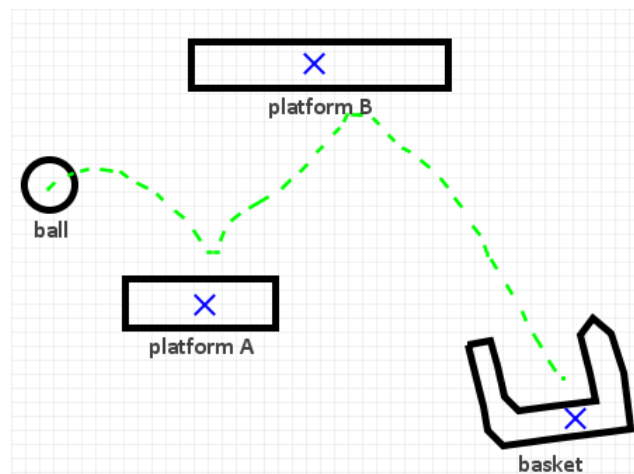


Figure 2-16: The user describes the same behavior as in 2-15.

This is motivated by the fact that when the user demonstrates a device's behavior, the resulting animation may not be physically correct. For example, consider the behavior in Figure 2-16. Since the user is free to describe movement and contact, the ball does not fall along a parabolic trajectory and the angles at which the ball bounces off the platforms are not symmetric. In addition, the timing of events may be imprecise: the ball 'sticks' to the platforms momentarily instead of bouncing.

If the user presses the SIMULATE button, PhysInk is able to produce a physically-correct simulation of the original behavior, shown in Figure 2-17. The simulation is qualitatively equivalent to the original behavior, according to the standards we have discussed in this section: the events have the same qualitative geometry and occur in the same causal ordering. Although the exact trajectory of the ball is not identical,

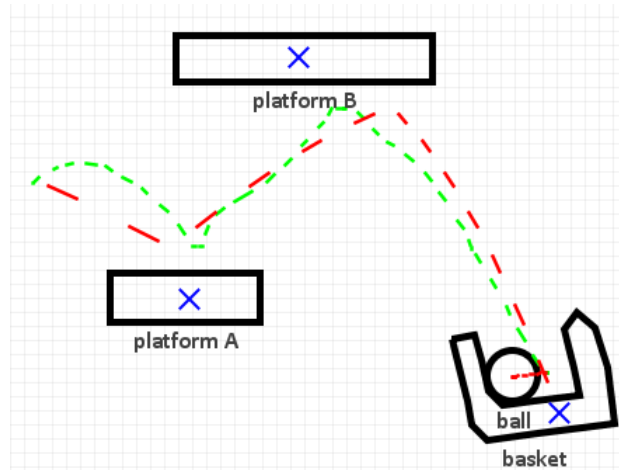


Figure 2-17: The user-described behavior (green) and the physically-realistic version of the behavior (red) found using PhysInk's SIMULATE button.

the user agrees that the simulation is an equivalent behavior. If the user has described behavior that is physically impossible, the system will inform the user and does not produce a new, equivalent behavior.

Chapter 3

Implementing PhysInk

In this chapter we explain PhysInk’s implementation, starting with an overview of its architecture, which integrates sketch and speech recognition, a physics engine, and a timeline to capture the user’s description. Next, we describe the processes used to capture structure in sketch mode, including stroke interpretation and speech-based labeling. We then describe the timeline and how it is used to capture behavior in demo mode. We explain how PhysInk assists the user in editing behavior by propagating changes through the timeline. Finally, we discuss how the timeline implements the behavior vocabulary described at the end of Chapter 2 and how it is used to find physically-realistic, equivalent versions of the user-described behavior.

3.1 Architecture Overview

The system’s architecture is illustrated in Figure 3-1. The Canvas is used to (a) convert sketch and speech interactions from the user into recognized shapes and text. In sketch mode, shapes are recognized and interpreted as objects or constraints based on their context, while text is used to label objects. The objects and constraints are (b) instantiated in the World, which is backed by the Box2D physics engine. The World contains the orientations, positions and velocities of objects in the ‘current’ frame of the user’s description, all of which is painted onto the Canvas (c). In demo mode, the user directly moves objects on the screen to demonstrate the device’s

behavior. At each frame, (d) the Timeline (a graph of movement and contact events) is updated with these manipulations. As the user plays back the Timeline, (e) the World is updated with the appropriate frame’s data, (f) which is then painted onto the Canvas.

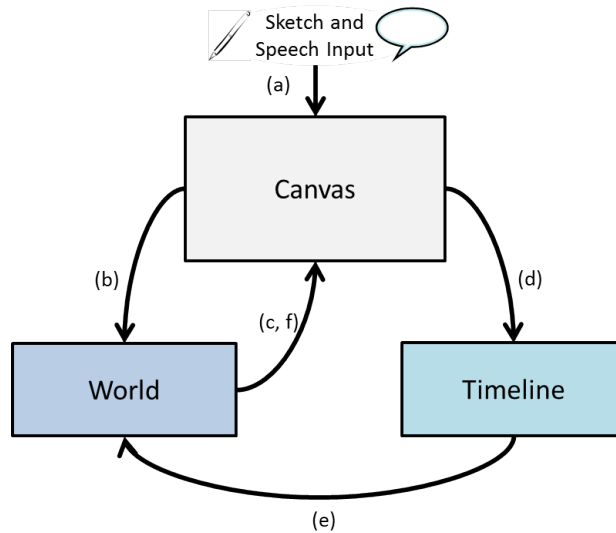


Figure 3-1: PhysInk architecture for capturing structure and behavior, consisting primarily of the Canvas, World and Timeline.

3.2 Understanding Structure

In sketch mode, the user’s sketch and speech are interpreted as an interactive, labeled diagram of physical structure. Figure 3-2 outlines the steps involved in this interpretation process. First, the user’s stroke is recognized as a geometric primitive by the Canvas and redrawn cleanly. This primitive is interpreted as either a physical object or a constraint, based on its shape and context. The interpretation is then instantiated in the World. In parallel, speech is converted to text and used to label objects in the sketch.

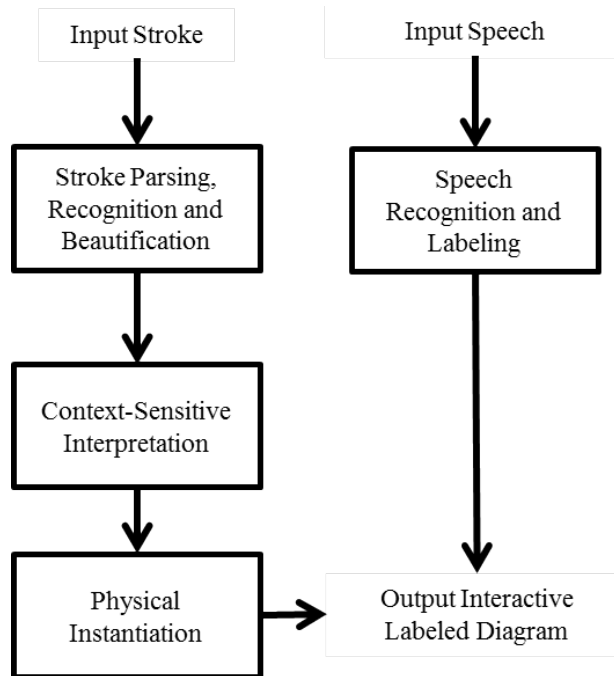


Figure 3-2: Processes involved in capturing structure.

3.2.1 Stroke Recognition and Beautification

When the user begins drawing on the canvas, a stroke object is created containing a sequence of XY screen coordinates defined by the pen’s movement. While the user is sketching, the stroke is painted onto the canvas in red. Once the pen is lifted from the canvas, the system begins the shape recognition and beautification process. Strokes with too few points are ignored, in order to handle accidental sketching.

Valid strokes are recognized and beautified using our implementation of the PaleoSketch system. PaleoSketch involves running separate, heuristic-based tests for a set of geometric primitives - in our case, lines, polylines, curves, ellipses, circles, helixes. If a primitive is recognized, PaleoSketch produces a beautified version of the stroke, which is used to replace the user’s points on-screen with a cleaner, finished version. After all the tests are completed, candidate geometric primitives are ordered by likelihood of a match with the stroke, again based on certain heuristics. The most likely primitive is used by PhysInk in the following steps.

Before the primitive can be interpreted as an object or constraint, two special cases must be handled. First, it is more likely that the user sketches rectangular

boxes than other quadrilaterals. PaleoSketch does not include a box recognizer; the closest it comes is recognizing something as a polyline, which in turn is composed of lines. We created a box recognizer that examines the lines making up a polyline and tests pairs of them for co-located endpoints, as well as perpendicular and parallel geometry.

The other special case is when the user wants to create a fixed joint using an X-symbol. PaleoSketch supports single-stroke line recognition, but X-symbols consist of two small, roughly perpendicular strokes. Therefore, when PaleoSketch recognizes two consecutive lines, they are passed to an X-symbol recognizer. The recognizer checks that the lines (a) intersect with each other, (b) have a small distance between their midpoints, and (c) are roughly perpendicular to each other. If the test is successful, a beautified X-symbol is passed to the interpretation process.

3.2.2 Context-Sensitive Interpretation

Once a stroke has been recognized as a geometric primitive and beautified, PhysInk interprets the result as a new physical object or constraint, depending on the primitive's shape and context. Here we explain the context-sensitive decision making process, which reflects the mappings in 2.1 in Chapter 2.

1. If the primitive is a line and it passes through a body (midpoint contained inside, and endpoints not contained inside), then the body should be deleted from the canvas.
2. If the primitive is a circle, ellipse or curve¹ inside an existing object, then a rotational joint should be created between the object and the object underneath it (another body or the background).
3. If the primitive is an X-symbol inside an existing object, then a fixation joint should be created between the object and the object underneath it (another body or the background).

¹Small circles are sometimes mistaken as curves by our implementation of PaleoSketch. Since it is highly unlikely that the user would create a small rope contained within an existing body, we interpret these types of curves as rotational joints.

4. If the shape is a helix and its endpoints are contained in at least one other fixed object, then a spring should be created with its endpoints tethered appropriately.
5. If the shape is a polyline or box, or an ellipse/circle not inside another object, then a rigid body should be created with the beautified shape.
6. If the shape is a curve not contained in another object, then a rope should be created with its endpoints tethered appropriately.

3.2.3 Physical Instantiation

Once an interpretation is determined, PhysInk instantiates the new object or constraint in the World, which is backed by the Box2D physics engine ². We first give an introduction to Box2D and its capabilities, before explaining how the beautified shape is instantiated in the physics world as a component or constraint.

Box2D: A 2D Physics Engine

Box2D is an open source physics engine for simulating the movement and contact of rigid bodies in 2D [7]. The Box2D API allows users to instantiate rigid bodies in the world. Each body is associated with a fixture, which contains information about the body's shape, density, coefficient of friction and coefficient of restitution (measure of the body's elasticity during collisions). Also, various types of joints between bodies can be created using the API, including weld (fixed), revolute (rotational), prismatic (translation along one axis only) and mouse joints. Mouse joints are created between an object and the user's mouse, which has the effect of creating a force on the object in the direction of the mouse - almost like a magnet.

Box2D's world maintains the instantaneous state of all physical bodies and joints, in terms of their static properties (mass, fixtures) and dynamic properties (position, orientation, velocity and force). The world can be 'stepped forward' by a specific amount of time in order to simulate the movement of bodies under the force of gravity,

²PhysInk is actually backed by jBox2D, a Java port of the Box2D physics engine.

mouse joints or other explicitly created forces. Gravity can be easily turned on and off.

During this 'step', objects can move only within the laws of physics, meaning that rigid bodies cannot overlap and are subject to the constraints imposed by joints. A contact listener is also associated with the world, which reports the beginning and end of contact events that occur during steps.

Components

When a new PhysInk object needs to be created, a new body is instantiated in the world. The beautified primitive is used as the fixture's shape, while its coefficients of friction and restitution are set to defaults that we have determined experimentally³. The mass of any body is set based on the area of its shape, using a constant density.

When creating new physical objects, there are two special cases where a single entity is represented internally as a set of bodies: ropes and non-rectangular polygons. In PhysInk, all bodies representing a physical entity are grouped into data structures called components. For example, in the case of instantiating curves as ropes, Box2D does not have a physical primitive with the appropriate deformability properties. Instead, PhysInk approximates a rope as a component consisting of many small, thin links. The chain is held together by rotational joints so that it deforms as a rope would.

Closed polylines that are not rectangular boxes must be handled similarly. Depending on the complexity of a physical body's shape, collision detection can be problematic for Box2D. Therefore, polylines are subdivided into many triangle-shaped fix-

³PhysInk can handle a variety of Rube Goldberg scenarios, as will be demonstrated in Chapter 4. Most involve balls that bounce off rectangular solids, roll down ramps, or land and remain in containers. These behaviors are all determined by restitution, a measure of the elasticity in the collision, which is taken by Box2D as the highest value among involved fixtures. We therefore choose high restitution (1.0) for rectangular solids, low restitution for freely-shaped solids (0.2), and medium restitution for circular solids (0.5). This way, ball-platform collisions are entirely elastic and ball-ramp or ball-basket collisions are fairly inelastic. Friction is taken by Box2D as the average coefficient of friction of objects in contact. In order to support the behaviors described above, we choose high friction for rectangular and circular solids (1.0) to prevent slipping between them, and low friction for freely-shaped solids (0.0) to allow balls to slide down ramps.

tures, using the poly2tri triangulation library⁴. Although the outline of the resulting component will be the original polygon, Box2D’s world only has to process collisions between triangles that are glued together, rather than an N-sided polygon.

Constraints

When a constraint needs to be applied to an existing component, a new Box2D joint is created in the world. Weld joints are used for fixed joints and revolute joints are used for rotational joints. Anchor points are given by the X-symbol’s or circle’s location. PhysInk’s canvas always includes a special invisible background body, which is used to anchor components that should be attached to the background.

Mechanisms that involve springs do not usually rely on translation along the spring’s perpendicular axis. Therefore, to simplify the user’s task, springs are allowed to vibrate only along a single horizontal or vertical dimension. To enforce this restriction, a prismatic joint is created between the bodies anchoring the spring, which allows one-dimensional movement only. Since Box2D does not provide a configurable spring, we implement our own primitive that obeys Hooke’s Law. At each step, the spring’s force (F) on connected components is recalculated based on deviation from the natural length ($l - l_0$) and the spring constant (k), where l_0 and k are configurable.

When a mechanism involves ropes, attaching their endpoints to components can be tedious for the user if X-symbols need to be drawn repeatedly to create fixed joints. For this reason, whenever a new component or rope is created, the canvas is scanned for any new overlaps between rope endpoints and components. If any exist, new fixation joints are created automatically.

3.2.4 Adjusting Structure

As described in Chapter 2, the user can adjust the positions and orientations of components in sketch mode. After selecting and dwelling on a component with the stylus, the component can be freely manipulated and moved in and out of contact

⁴poly2tri is an open-source 2D Delaunay triangulation library [3], which decomposes a polygonal area into a set of triangles. Its implementation is based on [10].

with other components.

Take, for example, the case where the user wants to place a ball directly on top of a platform. First, a mouse joint is created between the ball and the user’s pen point, resulting in a force on the ball in the direction of the pen. The world is stepped forward at the frame rate, allowing the component to move towards the user’s pen. While the user is manipulating the ball, all other components are fixed to the background, so that the user does not unintentionally change their initial positions. If the ball overlaps with the platform, the physics engine recognizes and resolves this physical impossibility, instead ensuring that the ball is flush against the platform.

3.2.5 Labeling Components

The user can label components in their sketch of structure using simple speech commands. For example, while the user is sketching the egg cracker’s ball they state out loud: `label ball`, which is then applied to the diagram. PhysInk converts speech into text using CMU Sphinx, an open-source toolkit for speech recognition [1]. The toolkit accepts a grammar⁵, which defines the phrases that should be recognized and returned by the speech recognizer. In order to support the labeling command, the grammar takes the following form:

```
<label_phrase> = label <component>;  
<component> = ball | platform | knife | paddle | rope | spring;
```

By default, the `component` variable contains a list of common physical objects. The user can also provide a text file with a list of component names that should be added to the grammar. Alternatively, the user can label structure by tapping a component with the pen and filling out a text entry box. The label is then added to the grammar, so that speech commands referencing the component are recognized.

Sphinx returns a string that is parsed for the label keyword. If the command is uttered while or immediately after a component is sketched, the label is applied to

⁵CMU Sphinx expects a Java Speech Grammar Formatted (JSGF) grammar.

the new component. Otherwise, the user must hover over or click on a component with the pen before uttering the label.

3.3 Understanding Behavior

In demo mode, the user can directly manipulate components to describe physical behavior. While the system can record these interactions and play them back verbatim, PhysInk attempts to build a deeper understanding of the user's description. In this section, we describe the timeline and its events, which are used to capture behavior. We explain how the user's manipulations are interpreted as movement, contact and end-contact events, and how the timeline can be used to assist the user in editing their description. Finally, we explain the processes involved in labeling events with the user's speech.

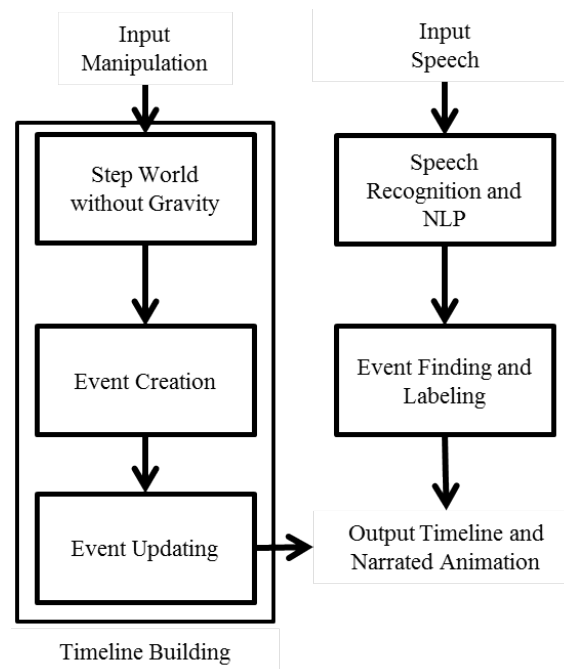


Figure 3-3: Processes involved in capturing behavior.

3.3.1 The Timeline

In a conversation between two humans, behavior is not thought of as pixels moving on a screen, but instead as a sequence of distinct events involving physical objects. An important part of this understanding is not only knowing which events comprise the behavior, but also knowing that one event causes another. Furthermore, an event's qualitative geometry (ball landing 'inside' the frying pan) is more important than its exact, numerical details (ball hitting coordinate XY in the frying pan).

As discussed in section 2.4, these observations form a vocabulary for describing behavior in software systems, which guides the structure of PhysInk's timeline and events. The timeline is a directed acyclic graph, where nodes can be four types of events (start, movement, contact and end-contact) and edges represent causality. The graph is acyclic, because events cannot cause themselves.

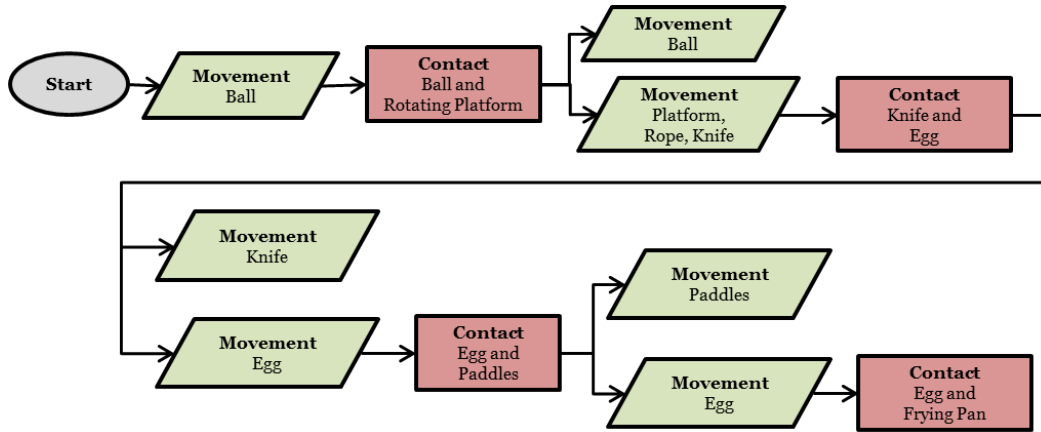


Figure 3-4: Timeline capturing the egg cracker's behavior.

As an introduction, consider the case of the egg cracker (Figure 3-4). The timeline begins with the start event, which represents an exogenous force that causes the ball's movement. The ball's movement causes a contact event with the rotating platform, which in turn results in the movement of both the ball and the platform. Since the rotating platform is connected to the rope and the knife, both the rope and the knife begin to move as well. This causes a contact event between the knife and egg, which results in new movement events for the knife and egg. The egg hits the paddles, causing another set of contact and movement events, leading to the egg falling into

the frying pan - the final event in the timeline. The result is a representation that aligns closely with how the user might explain the egg cracker's behavior.

Start Event

We assume that any behavior being described in PhysInk must start due to some exogenous force, whether gravity or a user-defined force. For this reason, the timeline always begins with a start event, which is used to represent the exogenous force. It is the only type of event that does not need to be caused by another event.

Movement Events

A movement event (Figure 3-5) captures both the user-demonstrated trajectory and the qualitative geometry of a single component's motion. In descriptions supported by PhysInk, we assume that movement can only be caused by an exogenous force or contact with another object. Therefore, movement events must be driven by exactly one other event: the start event or a contact event.

The literal motion of the component is recorded in a trajectory. The trajectory consists of a list of frames, each containing the component's position, orientation and velocity at a specific point in the user's description.

In order to capture the event's qualitative geometry, the trajectory is also interpreted relative to other objects. Consider, for example, the ball's movement in Figure 3-6. PhysInk determines whether the ball's trajectory crosses into the areas above,

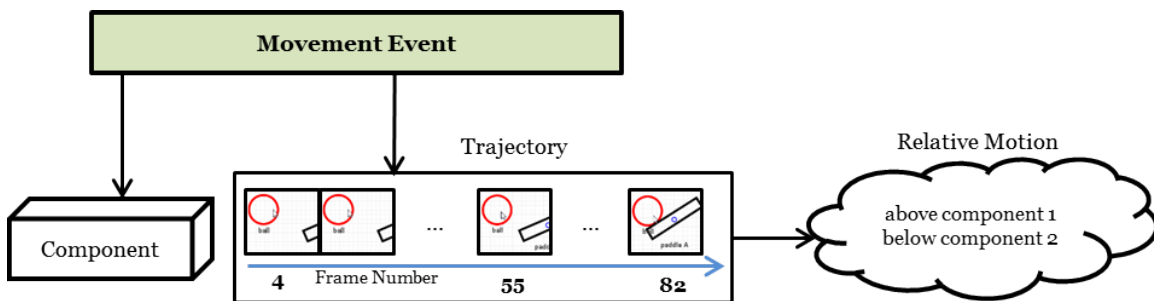


Figure 3-5: The movement event's data structure, which is associated with a component and a trajectory. The trajectory captures the component's motion as position, orientation and velocity in a list of frames.

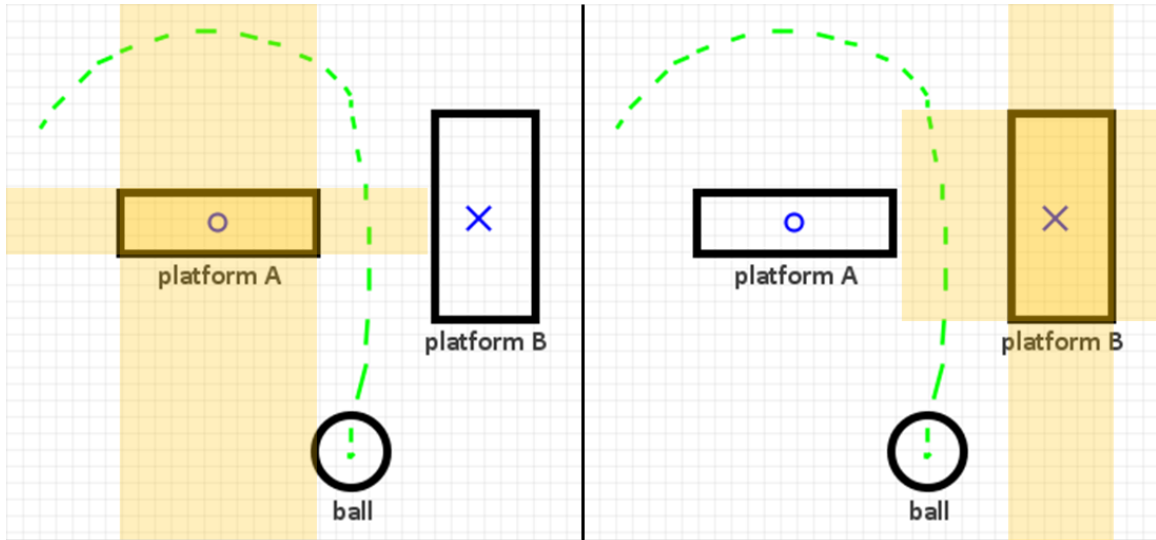


Figure 3-6: A movement event’s qualitative geometry is determined by checking crossings between a component’s trajectory and the above, below, left and right areas of other components. Here, the ball passes above and right of platform A, and left of platform B.

below, left or right of each nearby component (the platforms). Using this method, PhysInk determines that the ball passes above and right of platform A and left of platform B.

Contact Events

A contact event (Figure 3-7) records the components and exact coordinates of a collision. The contact’s qualitative geometry is also captured in terms of which surfaces collide with each other. For example, consider the contact illustrated in Figure 3-8 between the block and the paddle. While the exact coordinates of the contact point are recorded, the contact event infers that the bottom of the block and top of the paddle are in contact.

Contact events can be caused by the movement events of any components involved in the collision. However, note that because movement information is stored in movement events, the contact event does not store any motion - it is an instantaneous marker of contact between objects. Also, PhysInk assumes that any collision of

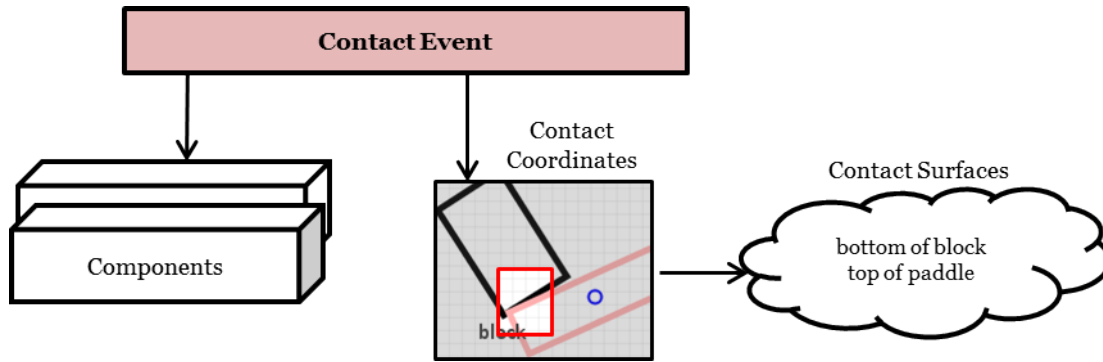


Figure 3-7: The contact event’s data structure captures exact collision data, as well as qualitative geometry in terms of which surfaces of components are involved.

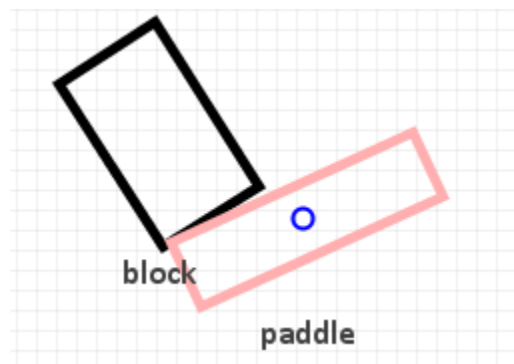


Figure 3-8: A contact event’s qualitative geometry is captured as the surfaces of components that are in contact. Here, the bottom of the block is in contact with the top of the paddle.

moveable⁶ components results in *more* movement. Accordingly, every contact event causes new movement events for each moveable component involved in the collision. This is explained further in the next section, where we describe the process of building the timeline.

End-contact Events

We have described how movement can be caused by either exogenous forces (represented by the start event) or contact, and how contact is caused by movement. While this vocabulary provides coverage of many physical behaviors, some mechanisms require that the timeline captures the *absence* of contact as a cause for movement.

Consider the mechanism demonstrated by the user in Figure 3-9. Initially, the

⁶Moveable components are any components not attached to the background through a fixed joint.

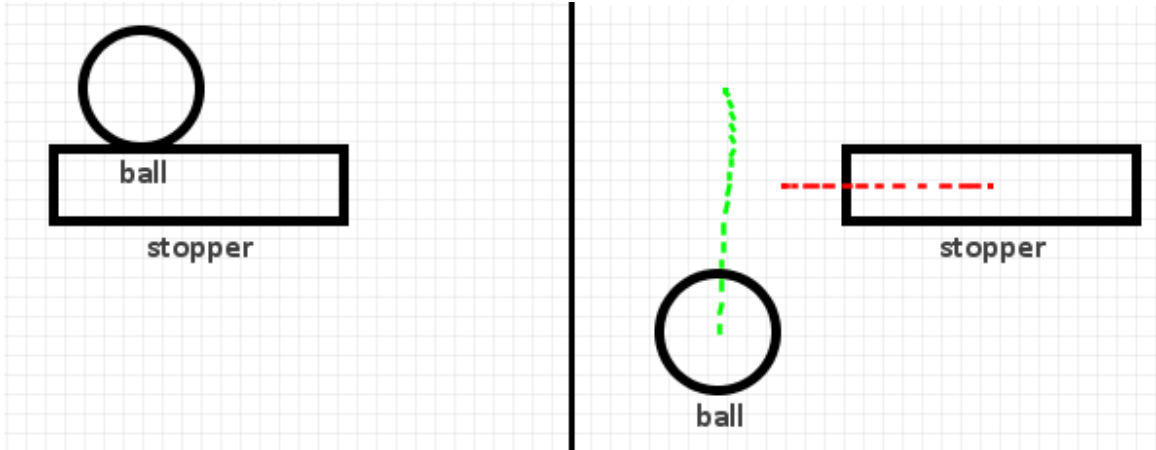


Figure 3-9: In this demonstration, a stopper impedes the movement of a ball. When the stopper moves out of the way, an end-contact event is created, resulting in the movement of both components.

ball rests on top of the stopper, in contact. When the user moves the stopper to the right, the absence of contact causes the ball to fall downwards. Therefore, to establish causality in the timeline, an end-contact event is used to mark the cause of the ball's movement.

In general, PhysInk checks a device's structure for two or more moveable components that begin in contact. During demo mode, if the user describes the movement of one of the components so they are no longer in contact, an end-contact event is inserted into the timeline.

3.3.2 Building the Timeline

We now describe how PhysInk populates the timeline with events, based on the user's sketch and speech manipulations. Recall the task of describing the behavior of a pinball machine, where a ball hits a series of rotating paddles (Figure 3-10). In the following subsections, we refer to Figure 3-11, which illustrates the timeline being populated as the user describes the machine.

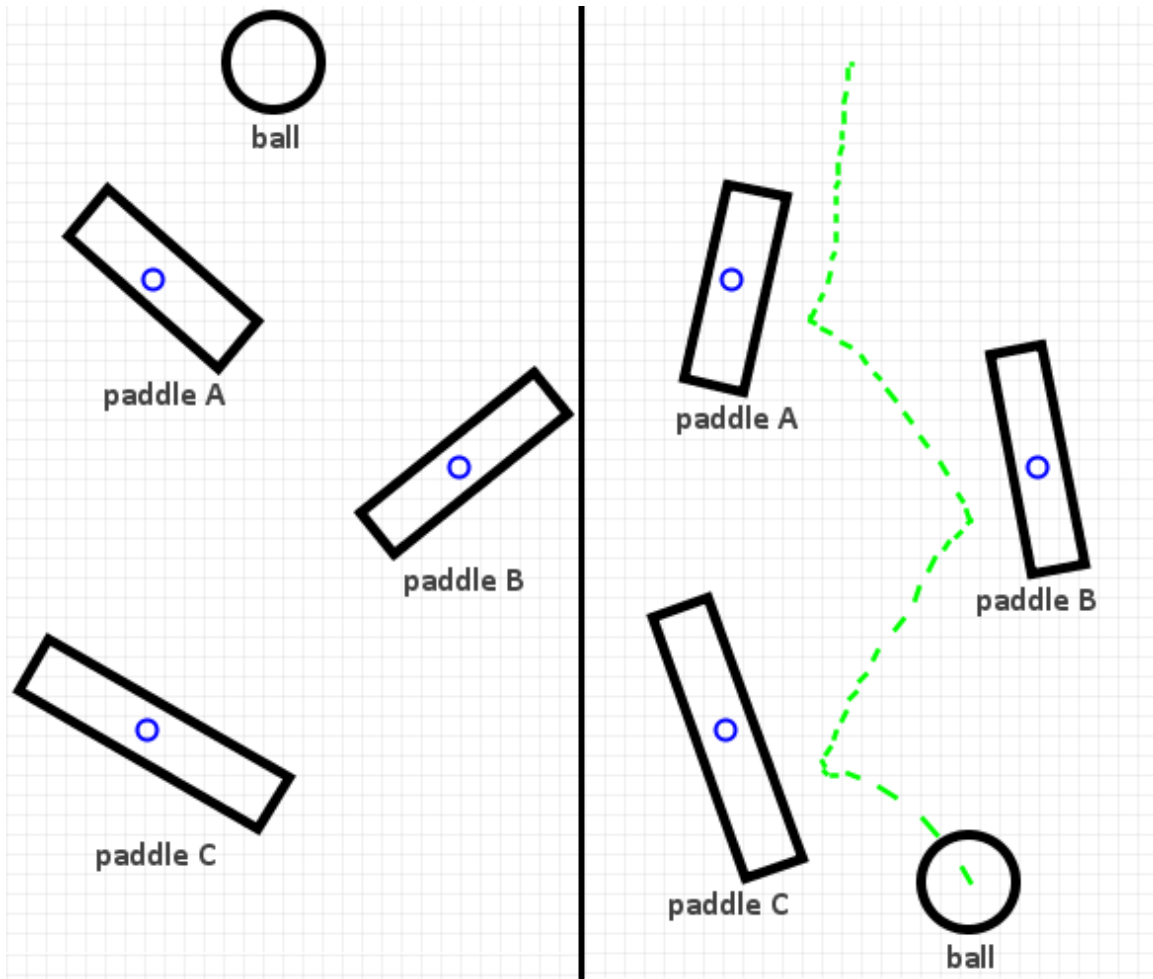


Figure 3-10: Demonstrating the pinball machine's behavior.

Enabling Manipulation

When the user begins manipulating the pinball in demo mode, the process used to capture behavior begins similarly to the one used for adjusting structure. First, a mouse joint is created to exert a force on the ball in the direction of the user's pen. While the pen is down, the physics engine is stepped forward at regular intervals, allowing the ball to move in response to movements of the user's pen. Note that in this case the other components are not fixed to the background, as they were in sketch mode. This means that when the ball makes contact with paddle A, the paddle moves in a physically-expected way.

Also, recall that PhysInk disables gravity during demo mode in order to allow the user to freely define the motion of sketched components. This means that as the user

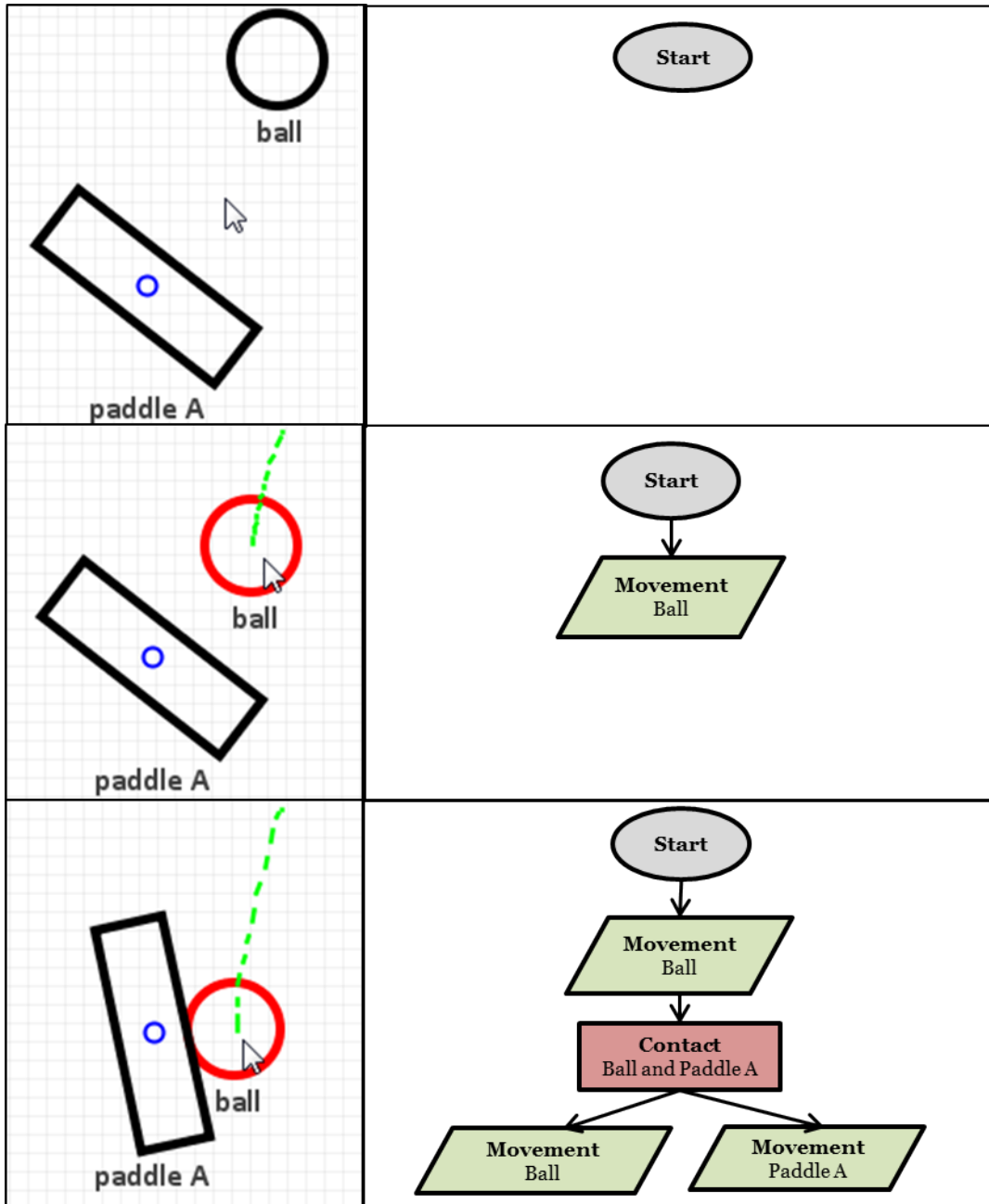


Figure 3-11: Building the pinball machine's timeline.

moves the ball to demonstrate the pinball machine, the paddles will not move unless they are manipulated by the user, either directly or indirectly through contact.

Creating and Updating Events

Initially, the timeline always begins with a start event, which represents an exogenous force (Fig. 3-11, top). When the user begins manipulating the ball, a movement event is added to the timeline (Fig. 3-11, middle). Since no contact events involving the ball have occurred at this point in time, it is assumed that the ball's movement is caused by an exogenous force (the start event).

As the user moves the ball towards paddle A, the world is stepped forward frame-by-frame. At each frame, the movement event's trajectory is updated with the ball's new position, orientation and velocity, as shown in Figure 3-12.

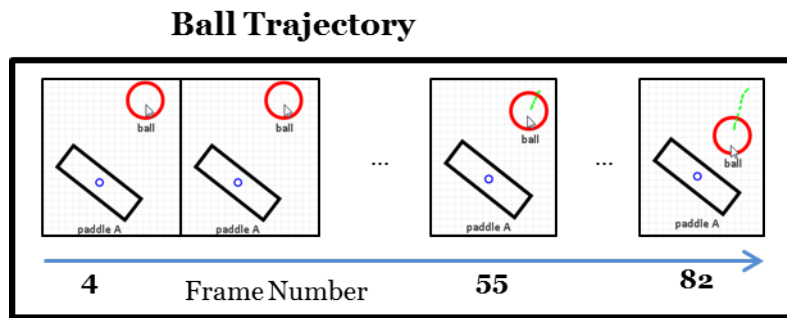


Figure 3-12: Updating the ball's trajectory with the user's manipulations.

When the ball hits paddle A, Box2D's contact listener reports the beginning of contact between the two bodies⁷. Consequently, PhysInk adds a new contact event to the timeline, and uses the ball's movement event as the cause (Fig. 3-11, bottom). Since both the ball and paddle A are moveable, contact between them implies that both of them move. PhysInk therefore creates two new movement events, both caused by the contact event, which are used to capture any future movement of the ball and paddle A that might be demonstrated by the user. The contact event acts as a marker, explaining the causal relationship between the ball's movement and the new trajectories of the ball and paddle A.

The user continues to move the ball to describe the pinball machine's behavior, making contact with paddles B and C. PhysInk captures these interactions by creating

⁷It is often the case that when the user demonstrates contact between two objects, Box2D's contact listener reports the same contact multiple times. This awkwardness is handled by ignoring reports of contact if the involved components have been in contact within the last 15 frames.

movement and contact events, building up the timeline shown in Figure 3-13. The timeline can be played back, rewind and saved as an animation of behavior.

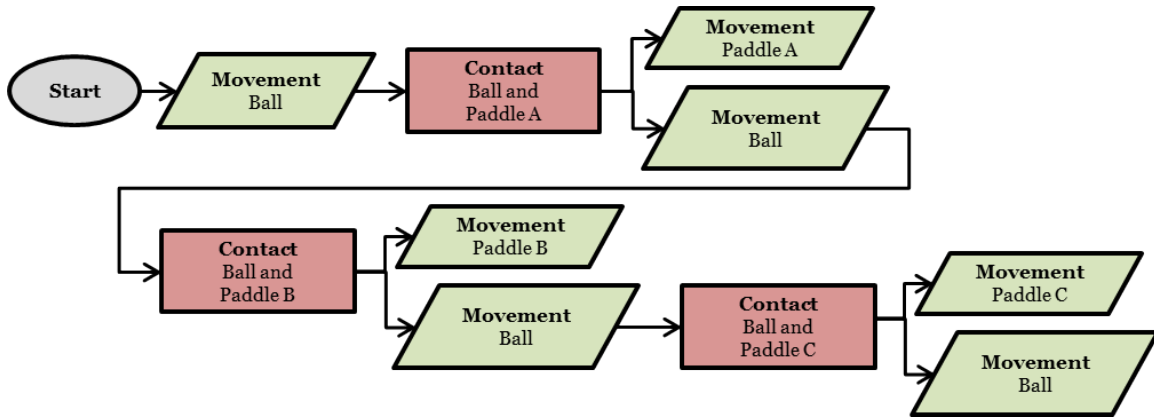


Figure 3-13: The pinball machine's timeline.

Capturing Concurrent Movement

Consider a modified version of the pinball machine, shown in Figure 3-14, where a second ball is introduced. Instead of bouncing off the paddles, the second ball falls

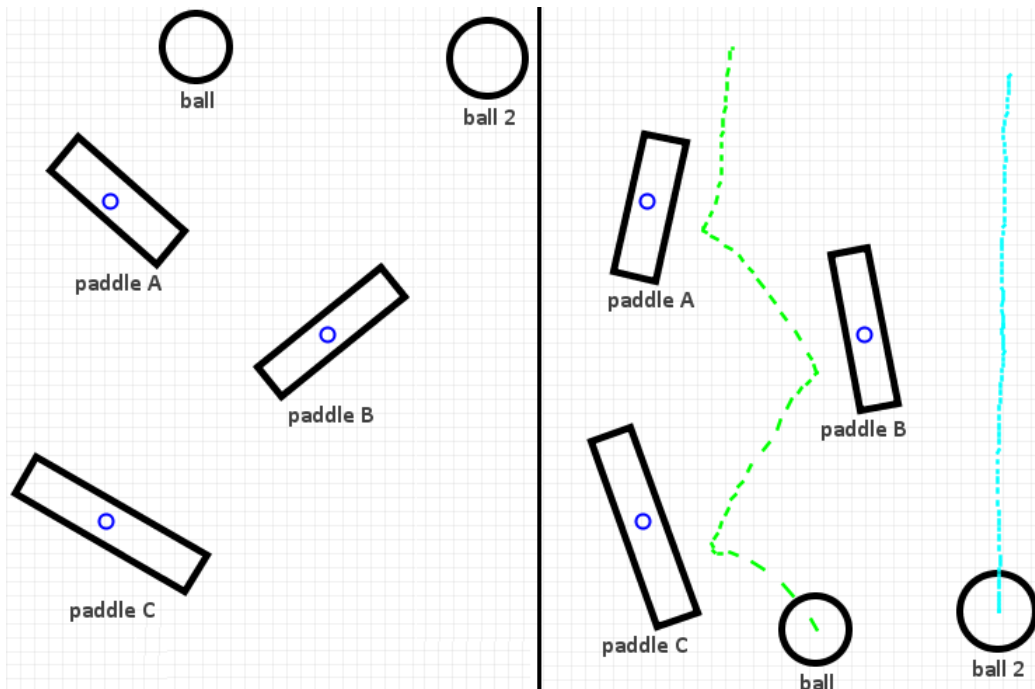


Figure 3-14: Modified version of the pinball machine, with a second ball that falls straight down.

straight down, unrelated to the original set of events. The user can demonstrate this by first recording the original ball's trajectory. Next, the user rewinds to the beginning and records the second ball's trajectory. While the second ball is moved, the original ball's behavior is played back concurrently from the timeline. In this way, the user can easily describe behavior that involves concurrent movement.

Assisted Editing

After describing the pinball machine, the user might decide that the behavior should be changed. For example, consider the edit described in section 2.3.5, where the ball hits paddle B on the right side of its pivot, causing the ball to bounce away without hitting paddle C.

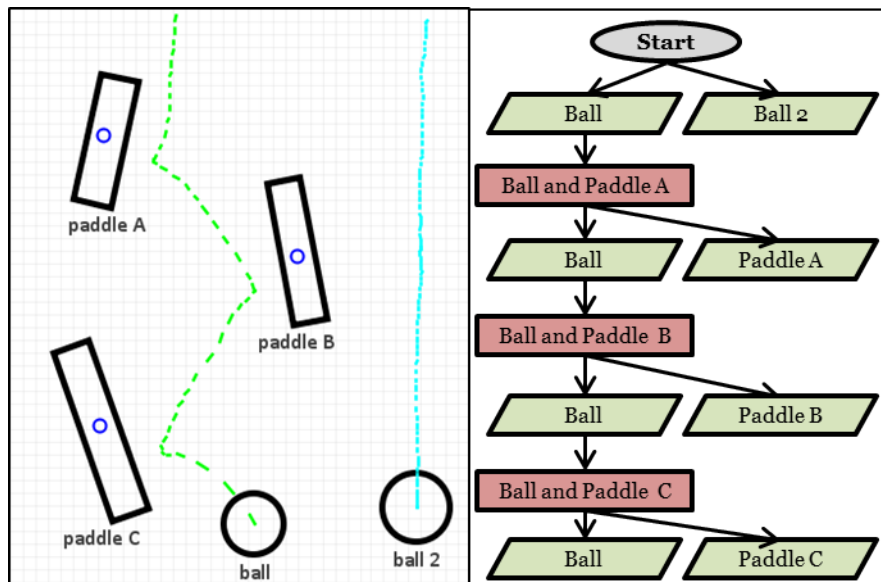


Figure 3-15: The canvas and timeline after describing the pinball machine's original behavior.

Figure 3-15 shows the timeline before the user has begun editing the behavior. At this point, the user rewinds the timeline to the point immediately after the ball has made contact with paddle A. He then moves the ball towards the right side of paddle B, in order to redefine the behavior.

Since the timeline captures causality, PhysInk is able to decide whether or not the user's edits affect events that follow. As soon as the user redefines the ball's

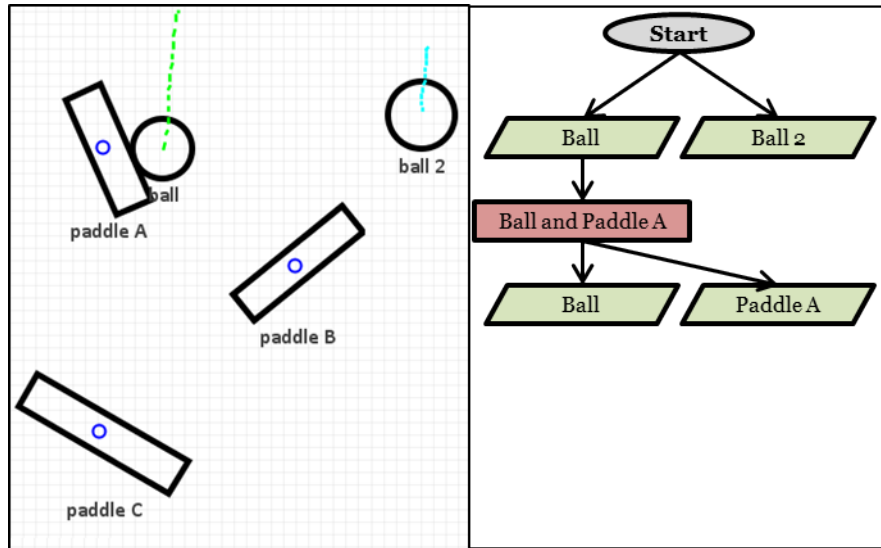


Figure 3-16: In order to edit behavior, the user redefines the ball's trajectory. Events previously caused by the ball's movement event are deleted from the timeline.

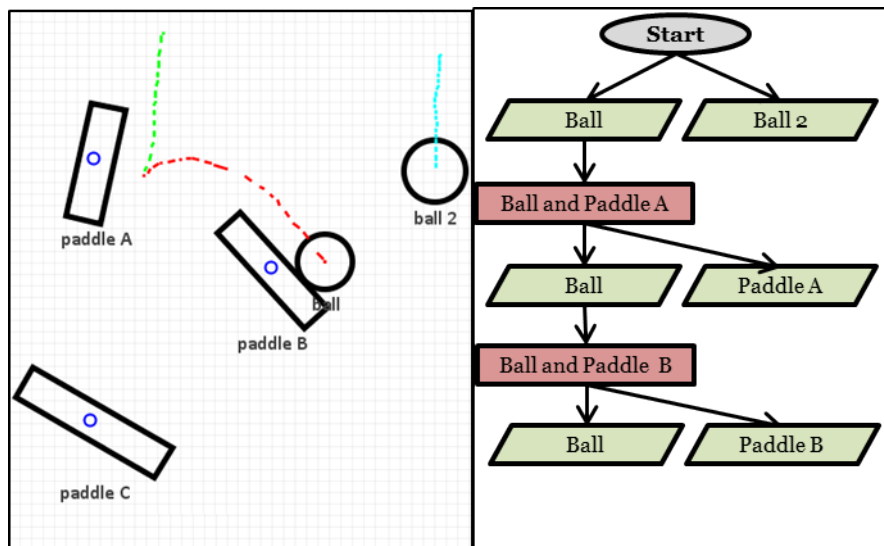


Figure 3-17: The ball hits paddle B and bounces away, resulting in the completed, new timeline.

trajectory, PhysInk reasons that all the events caused by its movement may no longer occur. The result is that the contact event with paddle B and all downstream events are deleted from the timeline, shown in Figure 3-16. Note that all events preceding the change are unaffected and that ball-2's movement, which was not caused by the original ball's movement event, is also left untouched. Now if the timeline were played back, neither paddle B nor paddle C would move, because the ball does not interact

with them.

The user continues to demonstrate the ball's new trajectory, by making contact with paddle B and bouncing away to the right. The timeline is rebuilt with a new contact event and set of movement events (Figure 3-17), based on the user's demonstration. The new behavior can be played back and is shown in Figure 3-18. Note that paddle C no longer moves, which logically follows from the user's new description. Also, since ball-2's motion had nothing to do with the user's changes, its movement event remains unchanged.

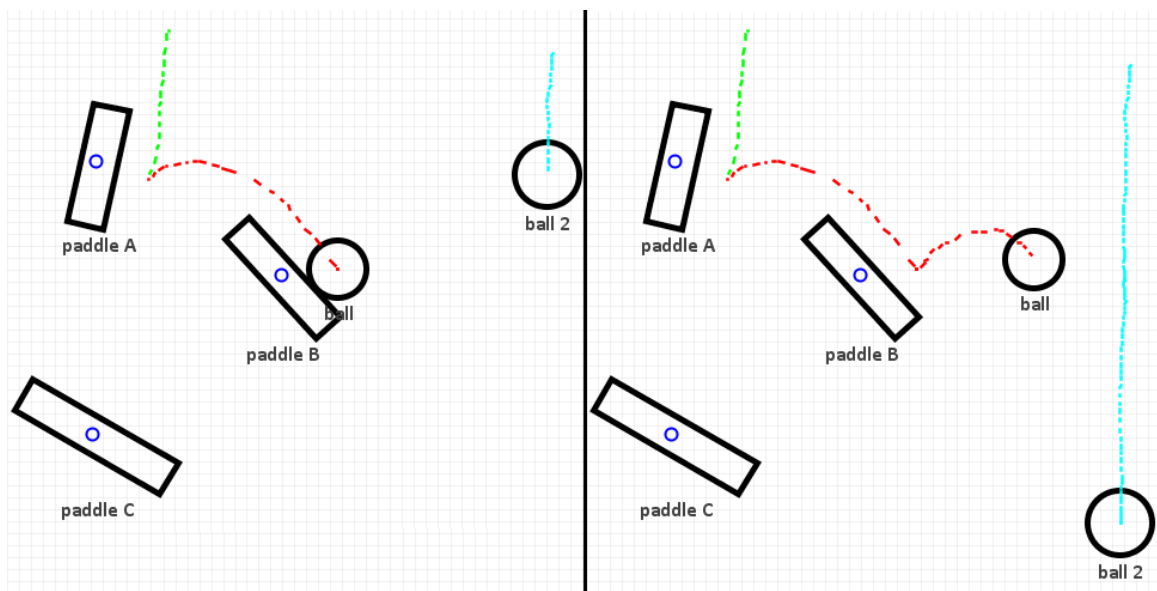


Figure 3-18: Editing the pinball machine's behavior.

3.3.3 Labeling Events

PhysInk is able to generate a narrative for the captured behavior by labeling events with default phrases, as described in section 2.3.3. Since the phrases are quite monotonous (`contact between ball and paddle`), the user can provide their own labels for events using simple speech commands. The commands can be uttered while an event is being demonstrated, or afterwards by pausing the animation near an event. Once again, CMU Sphinx is used to convert speech to text, where phrases must adhere to the following grammar:

```
<event_phrase> = <component> <verb> [<prep>] [<component>];  
<verb> = hits | falls | moves;  
<prep> = in | on | from | ...
```

`Component` is the same variable described earlier in the chapter and includes any user-defined labels for components. The `verb` and `prep` variables contain common verbs and prepositions used in Rube Goldberg machines, and can be extended by adding words to an input text file. Note that the second [`component`] value may or may not be spoken (e.g. `ball lands in basket` vs. `ball falls`).

Sphinx returns a string involving one or two component names. Next, the Stanford Parser⁸ is used to identify the subject, verb, preposition and object in the user's phrase. The timeline is searched for the temporally-nearest event that involves the subject and object components. If such an event is found, its default description is replaced with the user's phrase.

3.4 Finding Physically-Realistic Behavior

Since the user is explicitly demonstrating motion, the shape, speed and timing of movement and contact might not be physically-correct. In this section, we explain how PhysInk is able to find physically-realistic behavior that is qualitatively equivalent to what the user has described. This means that the events and order of events should match the original behavior, but the shape and speed of trajectories or the location of collisions need not be exactly the same. Apart from being a useful feature, this also demonstrates how well PhysInk has captured the user's understanding of behavior. If the user agrees that the simulation is equivalent to what they have described, it can be argued that the system 'understands' the behavior at a much deeper level than pixels moving on the screen.

⁸The Stanford Parser is a set of open-source Java implementations of probabilistic natural language parsers [12]. PhysInk uses the lexicalized probabilistic context-free grammar (PCFG) parser, which is based on [13].

3.4.1 Review: Qualitative Equivalence

In section 2.4, we introduced a vocabulary for describing behavior: events, their qualitative geometry and their causal relationships. The premise was that if two behaviors appear the same using this vocabulary, then they are also considered equivalent by the user. In section 3.3.1, we described the timeline and its events (start, movement, contact and end-contact) and how they are an implementation of the behavior vocabulary. The timeline captures events and their ordering in a causal graph. The events themselves capture the qualitative geometry of movement (motion relative to other objects) and contact (surfaces in contact). It follows that PhysInk can compare two timelines, each representing a specific physical behavior, and determine whether the behaviors are equivalent or not. The system's ability to find physically-realistic, equivalent behavior is centered on this idea.

3.4.2 The Simulator

We introduce one more part of PhysInk's architecture called the simulator, which is responsible for finding physically-correct, qualitatively equivalent behavior.

Since it is backed by the Box2D physics engine, the simulator is capable of generating many physically-correct simulations of the user's sketch. Simulations are entirely determined by the physical parameters of sketched components and constraints. For example, in the case of the pinball machine, varying the mass and friction of the ball and paddles will result in qualitatively different timelines. The difficult task of the simulator is to find the parameters resulting in a simulation that matches the user's description. This simulation will be physically-correct, because it is produced purely by the Box2D physics engine.

The simulator's method for performing this parameter search is summarized in Figure 3-19. First, initial values are chosen (seeded) for parameters that should be varied in the search for the simulated timeline. The device is then simulated under the force of gravity. During simulation, the movement and contact of components are captured in a timeline of events, which we call the 'simulated timeline'. The

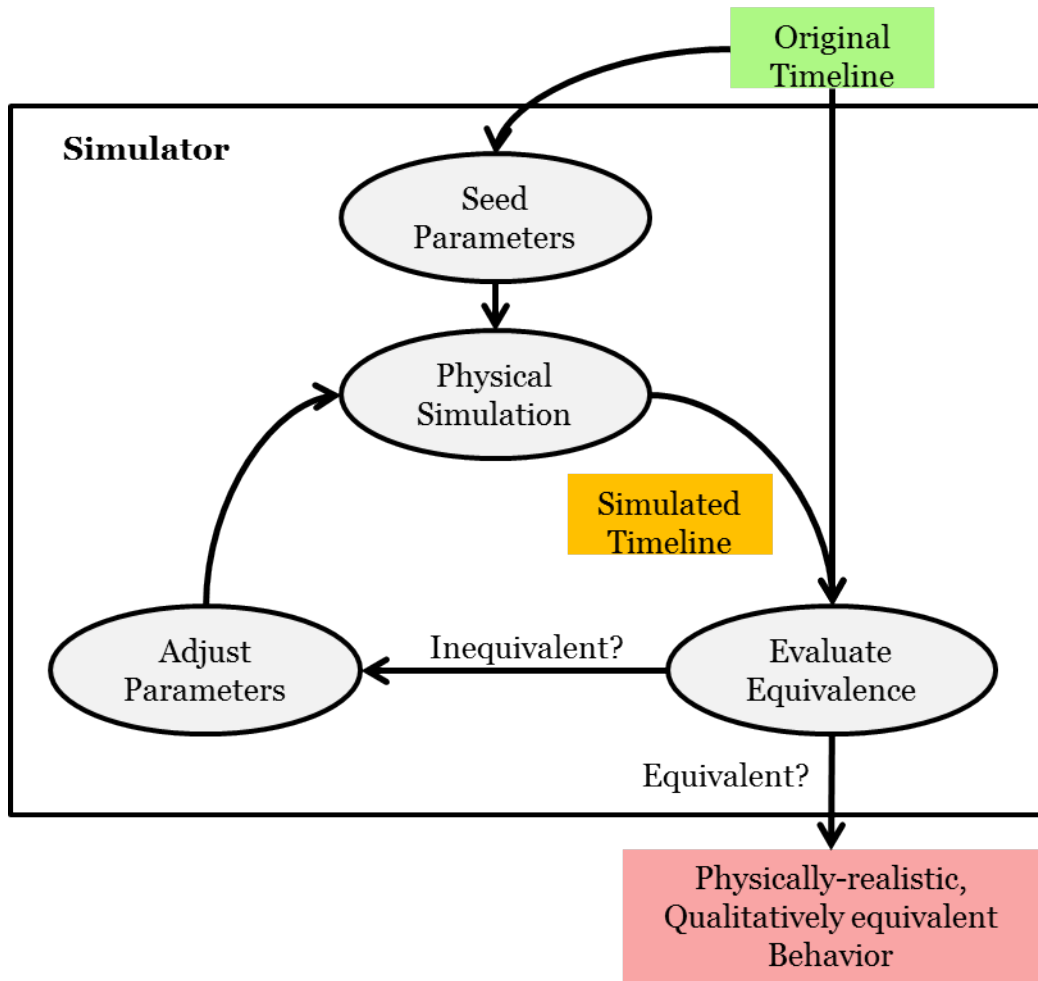


Figure 3-19: The steps followed by the simulator to find a physically-correct, qualitatively-equivalent version of the user-described behavior.

simulated timeline and original timeline are compared to determine whether or not they are qualitatively equivalent. The metric for qualitative equivalence is based on the behavior vocabulary described above and in section 2.4. If the simulation is not qualitatively equivalent, the search continues after adjusting the values of physical parameters driving the simulation. If the simulation is qualitatively equivalent, it is played back to the user as the physically-realistic version of the behavior they have described.

3.4.3 Seeding Parameters

Based on the user’s description of structure, a number of parameters are selected and seeded for the simulator’s search. The mass and friction of all moveable components (except ropes) are selected as free parameters. A component’s mass is seeded based on the area of its shape, using a constant density of $1.0 \frac{kg}{m^2}$. Its coefficient of friction is seeded with a value of either 1.0 (high friction) or 0.0 (low friction), as explained in section 3.2.3.

If a component’s initial movement is caused by an exogenous force (the start event), then its initial velocity is also selected for the parameter search. Its value is seeded using the direction and magnitude required to reach its initial destination. In Figure 3-20, for example, the ball’s initial velocity parameter is seeded based on the fact that it should make contact with paddle A. Note that the timeline’s notion of causality is being used here to provide a guess at the ball’s initial velocity.

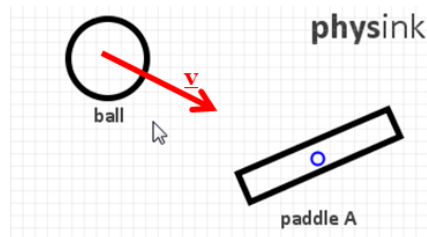


Figure 3-20: Seed vector for ball’s initial velocity.

Finally, for any springs, the natural length (l_0) and spring constant (k) are included in the search. The spring constant is always seeded with a value of 20.0, while the natural length is seeded with the spring’s sketched length.

Even for simple machines, the simulator faces a slow search through a multi-dimensional, continuous parameter space. For this reason, we make a few simplifying assumptions in order to make the space searchable. First, we assume that ropes are massless in the behaviors supported by PhysInk and as a result, their mass and friction values are excluded from the search. Also for simplicity, restitution (elasticity of collisions) is excluded from the search. Instead, a basic heuristic is used to choose constant, default values for components: elliptical and rectangular solids are given high restitution coefficients since they usually represent balls and hard platforms in

our use cases. Freely-shaped polygons are given low restitution coefficients since they usually involve ramps or concave containers off which objects should not bounce.

3.4.4 Evaluating Equivalence

Once the parameters are seeded, the search begins by generating a physically-realistic simulation resulting from the parameter values. The simulator must then evaluate whether or not the simulation is qualitatively equivalent to the user-described behavior.

As discussed earlier, the simulation is captured in a simulated timeline using the same processes described in section 3.3 to store the user’s description of behavior. The simulated timeline can then be compared with the original timeline to measure their qualitative similarity. This similarity is summarized by a qualitative and quantitative distance, where the qualitative distance is weighted more heavily.

The qualitative distance depends on the similarity of the events and their ordering in the timelines. First, do the same events exist in both timelines and do they occur in the same order? Next, event similarity is scored using qualitative geometry captured in movement and contact events: does the ball pass above the paddle B in both cases? Does the ball hit the top of the paddle A in both cases? Overall, the higher the mismatch between events or their ordering, the higher the qualitative distance between the timelines.

Although we are primarily concerned with the qualitative equivalence between the two timelines, we still want to find a timeline that most closely resembles the user’s description. Therefore, the simulator also calculates a quantitative distance by comparing the exact trajectory shape and contact points of matching events in the two timelines. The trajectories are normalized by their duration to account for differences in speed. Again, higher mismatches mean higher distances.

Throughout the search, the lowest combined qualitative and quantitative score and the parameters that resulted in that score are saved. Once the search is complete, the minimal-distance simulated timeline is played back as a demonstration of PhysInk’s understanding.

3.4.5 Adjusting Parameters

During each iteration of the parameter search, parameters are adjusted one at a time so that all combinations of values can be tried. Each parameter is searched over a particular range, which is discretized to reduce search's complexity. For example, if a component's initial velocity vector was seeded as Θ , then the search sweeps from $\Theta - 45^\circ$ to $\Theta + 45^\circ$ in 15° increments. If a spring's natural length was seeded as l_0 , the search sweeps from $\frac{l_0}{2}$ to $2l_0$ in $\frac{l_0}{4}$ increments.

In many cases, this results in a slow high-dimensional search. It is difficult to implement a hill-climbing algorithm since the relationship between parameter values and qualitative distances is highly nonlinear. Instead, the simulator halts the search early if the qualitative score of the simulated timeline crosses a certain threshold. Improving the speed and quality of the parameter search is subject for future work.

3.4.6 Examples

We now give two examples of the simulator's ability to find physically-correct behavior that is qualitatively equivalent to the user's description. On the left side of Figure 3-21, we see that the user has described the ball bouncing off platform B before landing in the basket. On the right side, we see the behavior found by PhysInk's simulator.

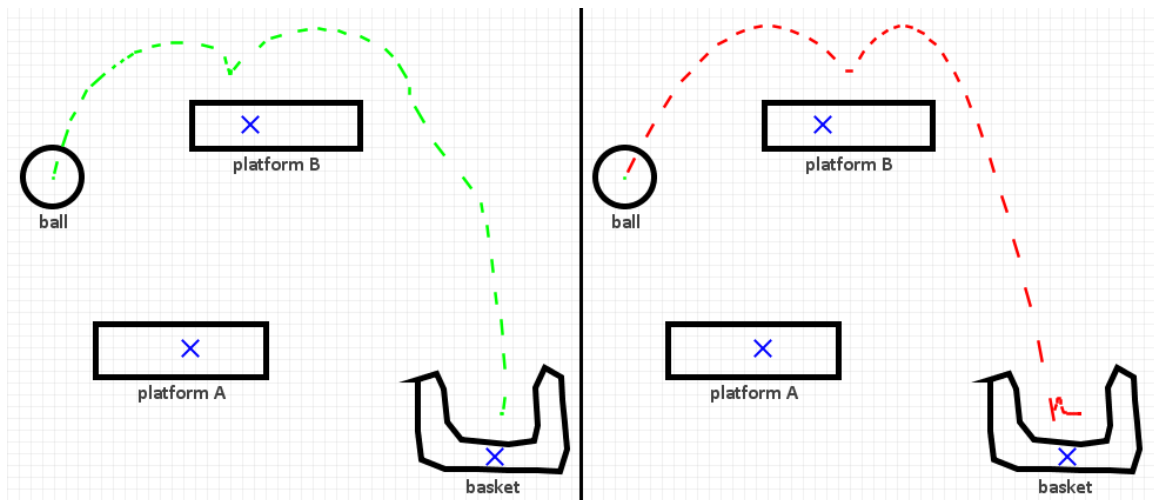


Figure 3-21: The user-described behavior (left in green) and the physically-realistic, qualitatively equivalent behavior found by PhysInk's simulator (right in red).

First, the simulation appears physically correct: the trajectory is parabolic and the angle of reflection off platform B is symmetrical. It is also qualitatively equivalent: the events and the order of the events are the same. Specifically, the ball-platform B contact event is equivalent because the qualitative geometry is the same: the ball bounces off the *top* of platform B. Furthermore, the ball lands *inside* the basket, instead of hitting any of its other surfaces and bouncing away.

Next, consider the modified behavior described on the left side of Figure 3-22, where the ball descends straight into the basket without touching any platforms. PhysInk finds a physically-realistic, qualitatively equivalent behavior shown on the right side of Figure 3-22. Its trajectory is clearly more realistic than what the user has demonstrated. The behavior is considered equivalent because the qualitative geometry of the ball’s movement is the same: the ball moves *above* platform A and *below* platform B. Once again the ball-basket contact event is equivalent because the ball hits the same surface in both the description and simulation.

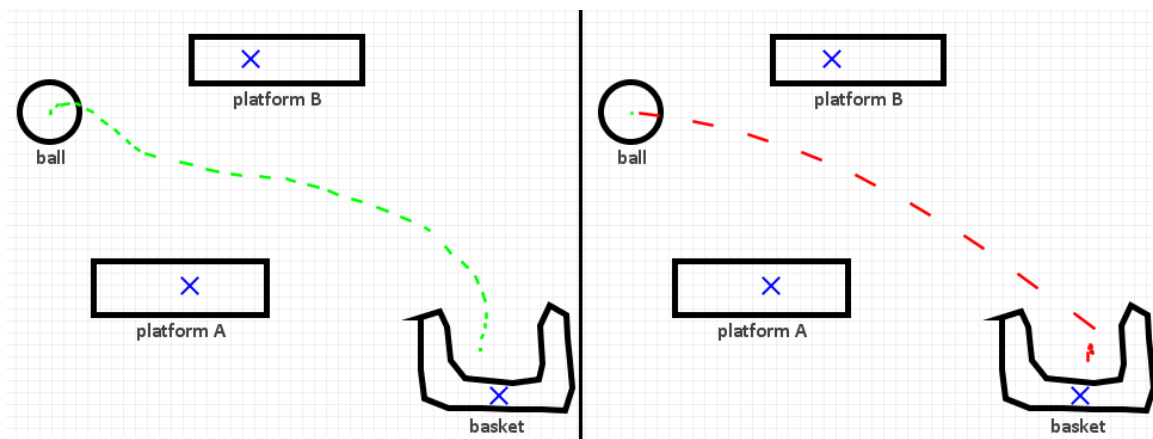


Figure 3-22: Modified behavior described by the user (left in green) and the physically-realistic, qualitatively equivalent behavior found by PhysInk’s simulator (right in red).

Although these are simple mechanisms, we see this feature as a proof-of-concept in the utility of timelines. We also see this feature as a way of validating the behavior vocabulary introduced in this thesis. We give more examples of finding physically-correct, equivalent behavior in the next chapter, where we present a portfolio of machines that can be described in PhysInk.

Chapter 4

A Portfolio of Machines

Throughout this thesis, we have presented the functionality and implementation of PhysInk primarily through two similar examples: the egg cracker and the pinball machine. In this chapter we demonstrate other types of machines whose structure and behavior can be captured using PhysInk.

4.1 Basketball Collision

Although this example was covered briefly in Chapter 2, we revisit it in more detail. The user wants to describe two basketball shots, where the balls collide in mid-air and fall into separate baskets. First, in sketch mode the user sketches the balls and baskets on the canvas, fixing the baskets to the background through fixed joints. Next, the user switches to demo mode (Figure 4-1, left) and captures ball-1's motion in a simple timeline: the start event causes the ball's movement event, which causes contact with basket 1. Note that the user describes ball-1's entire trajectory, including the change in direction that will be caused by the contact event with ball-2.

In order to record the concurrent movement of ball-2, the user rewinds the timeline to the first frame. As ball-2 is moved towards the collision location, ball-1's trajectory is played back concurrently using the timeline, making it easy to describe the collision (Figure 4-1, middle). The contact event causes ball-1's movement event to be split into two movement events: prior to the collision and after the collision. ball-2 also

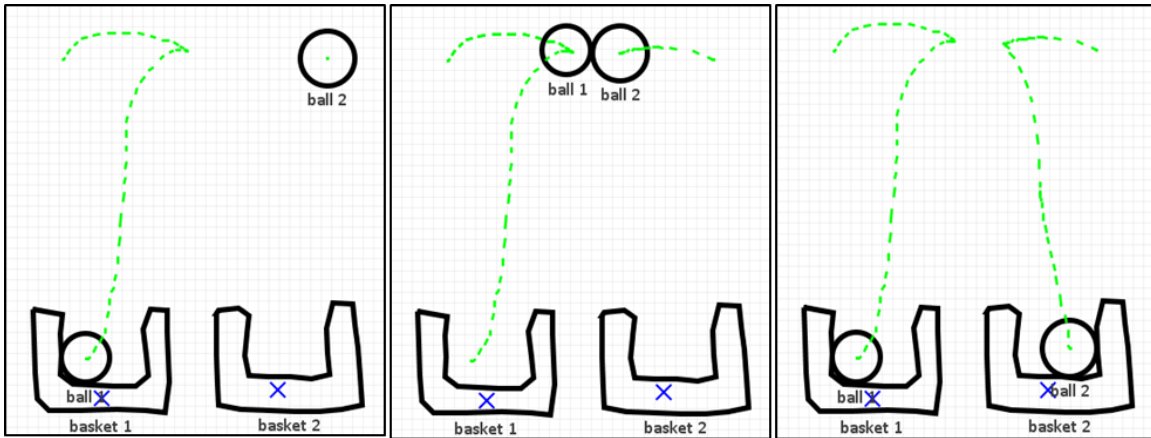


Figure 4-1: Describing concurrent motion and the collision of two basketballs.

has two movement events, separated by the collision. The post-collision movement events cause contact with the two baskets (Figure 4-1, right). The captured timeline is outlined in Figure 4-2.

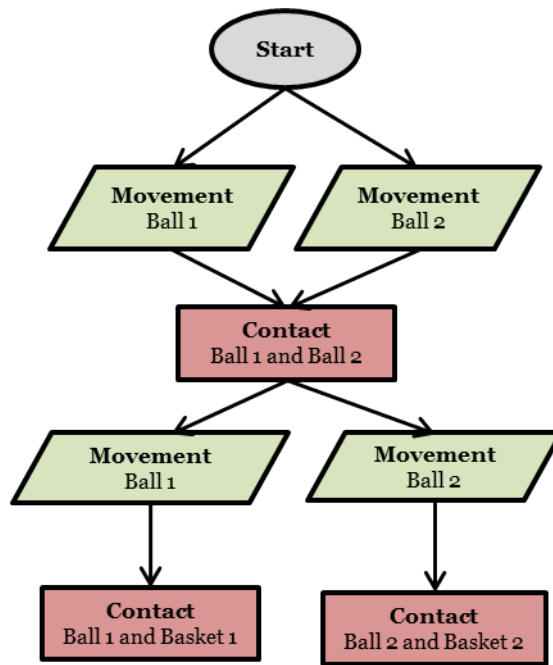


Figure 4-2: Timeline for the basketball collision.

The user can check that PhysInk has correctly captured the 2-ball collision by pressing the SIMULATE button to generate a physically-correct, qualitatively equivalent version of the behavior. PhysInk’s simulator proceeds to search for initial velocities, friction coefficients and masses of the two balls that result in a qualitatively

equivalent timeline, under the force of gravity.

Figure 4-3 shows the user’s description in green (left) and the physically-correct simulation generated by PhysInk in red (right). We note that the trajectories are quantitatively different, but in terms of events and causality, the two behaviors are equivalent. Small, sharp turns at the bottom of the simulated trajectories depict the balls rolling and bouncing before they settle into the baskets, a physically-realistic detail that is not captured in the user’s simplistic description.

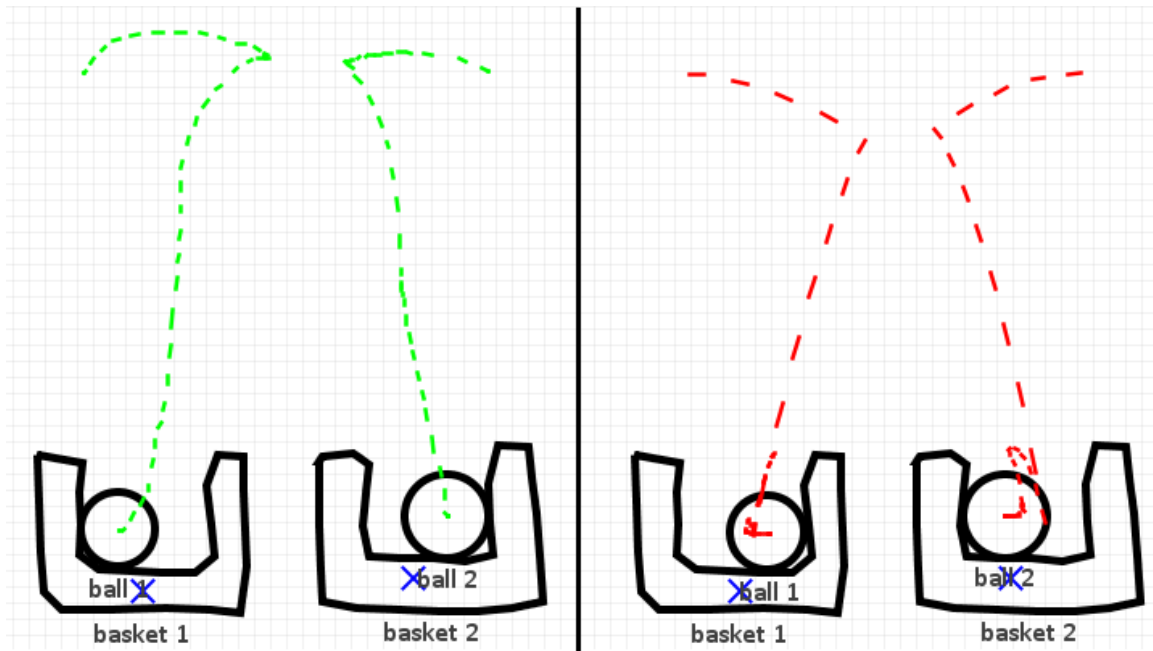


Figure 4-3: The 2-ball collision as described by the user (left in green) and the qualitatively-equivalent, physically-realistic simulation produced by PhysInk (right in red).

4.2 Driving Range Golf Ball Helper

In this example, the user describes the mechanism used in a driving range’s golf ball helper, which tees up a ball for the golfer. A spring-bound peg hits a mechanical arm, which in turn pushes a golf ball onto the tee.

The user sketches the device’s structure, using fixed joints to attach the various anchoring bodies to the background. A rotational joint is used to attach the arm

to the middle anchoring body. A spring is sketched, completing the peg's structure. PhysInk treats the sketched length of the spring as its natural length. The golf ball helper's structure is illustrated in Figure 4-4.

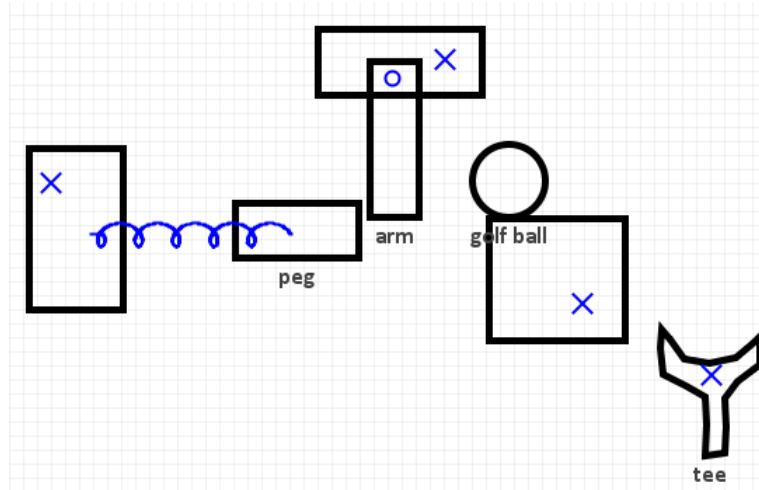


Figure 4-4: The golf ball helper's structure.

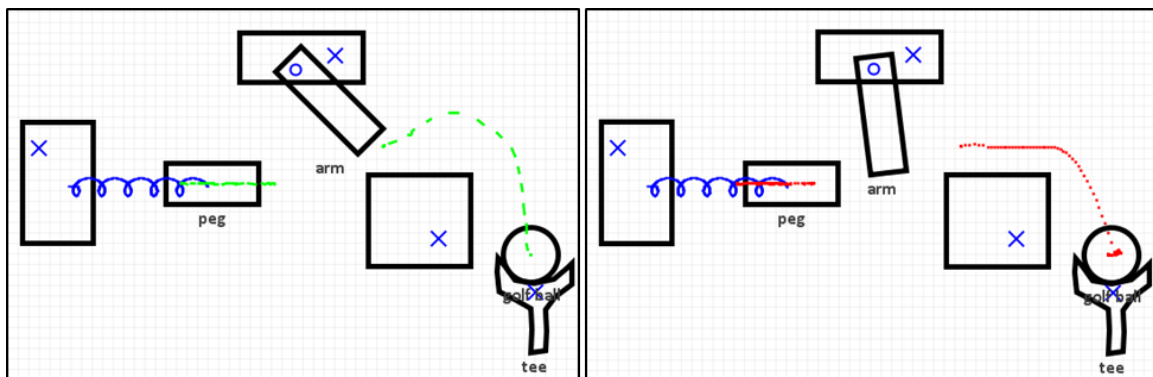


Figure 4-5: The user describes the golf ball helper's behavior in demo mode (left in green). PhysInk generates a physically-realistic, qualitatively equivalent version of the behavior (right in red).

In demo mode, the user manipulates the peg, arm and golf ball to demonstrate the device's behavior. The trajectories of the components are shown on the left side of Figure 4-5. Again, the SIMULATE button can be used to generate a qualitatively-equivalent physical simulation of the device. In addition to the golf ball's mass and the peg's initial velocity, the spring's natural length and constant are included as parameters in the simulator's search for an equivalent timeline. The results of the

search are shown on the right side of Figure 4-5.

Notice that the simulated peg oscillation (red horizontal line near the peg) is smaller than the user-described peg oscillation (green horizontal line near the peg). The reason behind this stems from the fact that the user has described the golf ball landing and remaining on the tee. If the peg oscillates too heavily (e.g. as heavily as it does in the user’s demo), the arm may push the golf ball so hard that it flies off the tee. The simulator has found parameters for the peg so it only moves as far as needed to impact the arm and, in turn, the golf ball. The nearly solid red line representing the ball’s trajectory indicates that it rolls slowly and lands lightly on the tee. This is a testament to how well the timeline captures the user’s description of behavior.

4.3 Cup Dispenser

In this final example, the user wants to describe a more complicated Rube Goldberg machine, where a cup is dispensed in a typically whimsical and complicated manner. First, a ball rolls down a ramp and then lands in the cup to be dispensed. The cup

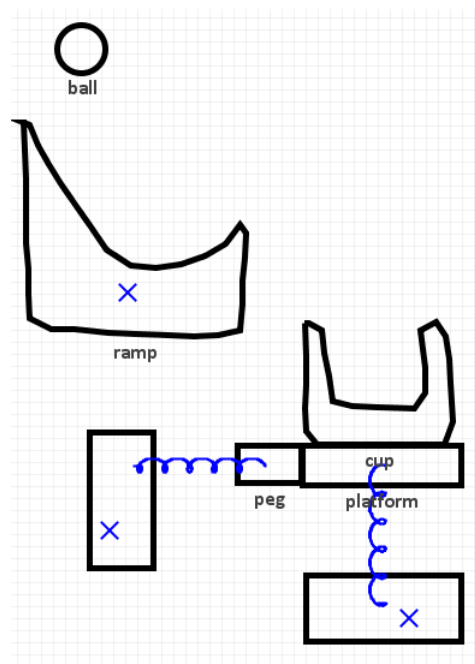


Figure 4-6: The cup dispenser’s structure.

sits on a spring-supported platform, which in turn gets weighed down by the ball. A peg presses on the platform from the left. Once the cup weighs the platform down enough, the peg is released and pushes the cup off the platform.

Despite its complexity, describing this machine is as easy as the other examples in PhysInk, by virtue of the interface's built-in physicality. It is important to note that the machine's behavior is dependent on the peg starting in contact with the platform. Similarly, the cup should sit flush on top of the platform. Since, at first, the dispenser's structure is sketched approximately, these objects may not be in contact with each other. Both of these structural details can be adjusted in sketch mode by selecting and dwelling on the component with the pen, and then moving it into contact with the platform.



Figure 4-7: Adjusting the cup dispenser's structure. The user selects the peg with the pen (left) and dwells until it turns dark green in color. The peg can then be moved (middle) so that it is flush against the platform and released to save the new configuration (right).

The cup dispenser's behavior is then described in demo mode by the user (Figure 4-8). The ball is moved down the ramp and weighs down the cup and platform. Once the peg is clear from the platform, the user manipulates the peg to push the cup and ball away from the machine.

The user presses the SIMULATE button to generate a physically-realistic version of the behavior. For this machine, it is important that the timeline has captured the causal link between the platform being weighed down and the peg's movement. When the user begins to manipulate the peg, PhysInk looks for the cause of this movement and can find no obvious contact event. In fact, it is the absence of contact that causes the peg's movement. In order to maintain a logical causal chain in the timeline, PhysInk creates an end-contact event for any unexplained movement of components that have recently ceased contact. If this were not done, the peg's movement would be attributed to an exogenous force. In simulation, this would result in unexplained,

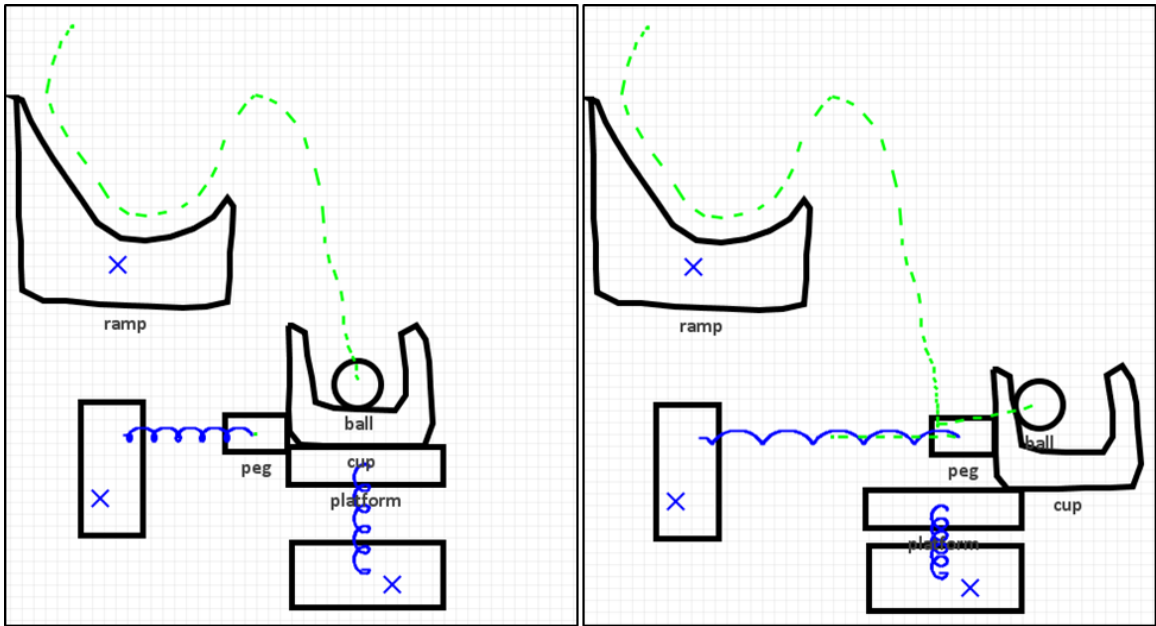


Figure 4-8: Describing the cup dispenser's behavior. The user describes the ball's trajectory, weighing down the cup and platform (left). The peg is then used to push the cup off the platform (right).

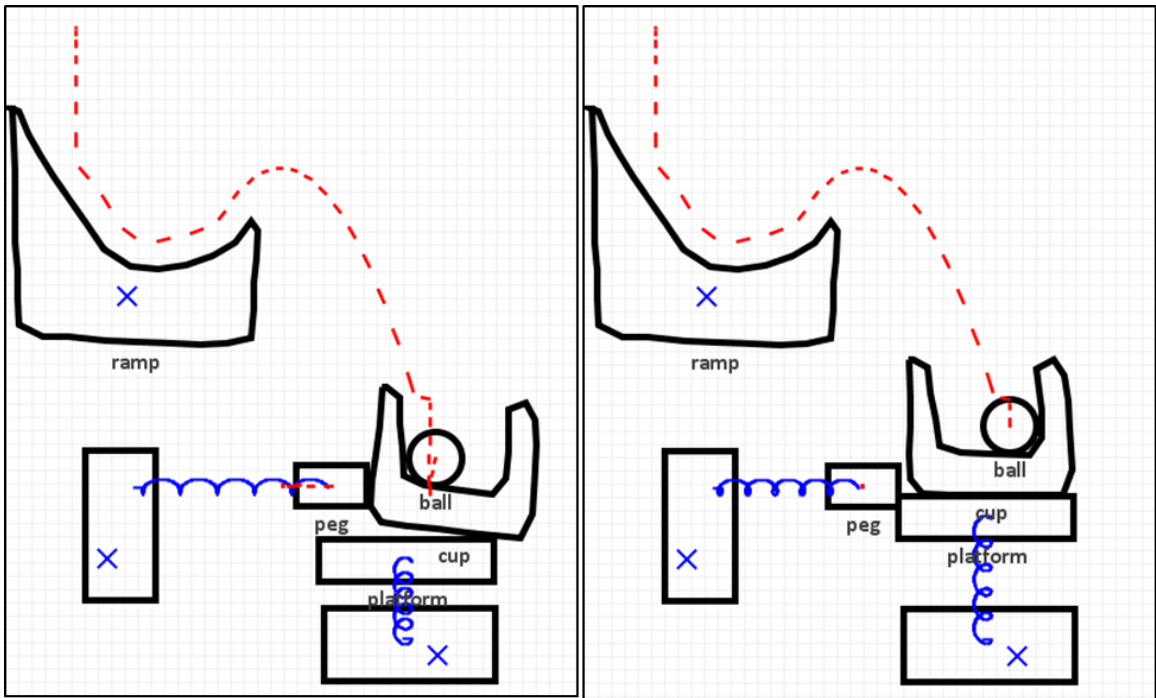


Figure 4-9: The physically-correct, qualitatively-equivalent simulation of the cup dispenser's ball and peg, shown in red. The peg does not move (left) until the ball has weighed down the platform enough so it does not impede its movement (right).

badly-timed movement of the peg, possibly producing a qualitatively inequivalent behavior. Instead, PhysInk produces the simulation shown in Figure 4-9.

Here we see that the peg does not move until the platform has moved out of the way. The simulator has increased the peg-spring's natural length parameter so that when the platform is present, the peg is compressed. When the platform is displaced by the ball's weight, the peg can return to its natural length, pushing the cup sideways. This successful search is a direct result of capturing the user's intended description of behavior in the timeline.

Chapter 5

Related Work

The work in this thesis was motivated by a lack of tools that support natural descriptions of physical behavior. In designing an intuitive user interface and the timeline, we integrated several ideas from related work on sketching structure, animation-by-demonstration and capturing behavior. In this chapter, we discuss prior research in these three areas, and how these ideas guided the design of PhysInk.

5.1 Systems for Capturing Structure

CAD tools like SolidWorks and AutoCAD are by far the most developed software systems for describing physical mechanisms. The user can create a 3D model of a device's structure primarily using a traditional windows, icons, menus, pointer (WIMP) interface, but also by sketching the device's basic contours on a tablet. The model can then be rendered with shading and colors to convey a high sense of realism. It can even be used in complex physical simulations to evaluate functionality, strength and safety. However, creating this model requires an advanced understanding of the device's structure, including dimensions, angles and materials. In contrast, our system targets the earlier stages of the design process, where the user wants to quickly explore function rather than form. Also, since most CAD tools' functionality is accessed through menus and icons, the learning curve is steep compared to PhysInk, where the user can freely sketch on the canvas to describe structure and behavior.

The ASSIST system [4] provided a much more natural way to explore a device’s structure, which was adopted in PhysInk: the user could sketch objects and constraints on a digital whiteboard, after which they were directly instantiated in a 2D physics world. The user could then press ‘GO’, at which point the user’s sketch would animate, playing out a simulation of the device’s behavior under the force of gravity.

Both ASSIST and PhysInk contrast with traditional CAD tools in that they are built for 2D sketches of simple mechanical devices, rather than complex, high-fidelity 3D designs. However, the intended usage model of ASSIST is still fundamentally different from that of PhysInk: the former supports descriptions of structure rather than behavior. In ASSIST, the focus is on describing form that implies function under the force of gravity. This means the user can sometimes enter a repetitive loop where they incrementally tweak structure and hope that the simulation matches their expectations. In PhysInk, the user demonstrates behavior a single time, which can be captured and used to guide a physically-realistic simulation of the device.

5.2 Animation-by-Demonstration

PhysInk’s user interface for describing behavior is inspired by K-Sketch [9], a general animation tool that allows the user to record motion by directly manipulating the sketch. We also adopt K-Sketch’s ability to record concurrent motion, by first recording one trajectory, then rewinding and recording the second trajectory while the first is played back simultaneously. Both of these fundamental UI features create a very natural, intuitive experience for the user when they are describing movement. In the domain of physical devices, PhysInk takes K-Sketch a step further through its ability to understand and enforce physical constraints. This relieves the burden on the user when they are demonstrating behavior, since objects move as they are expected to move, without needing to worry about overlaps or other physical impossibilities. Also, PhysInk’s timeline represents a departure from traditional animation, since motion is captured in distinct events instead of graphics in a series of frames.

5.3 Systems for Capturing Behavior

ASSISTANCE is an extension of ASSIST that focuses on behavioral descriptions and learning causality from sketch and speech [15]. This understanding of behavior is used to adjust the physical parameters in ASSIST’s 2D model of the device, in order to converge more quickly on the user’s intended behavior. The idea that an understanding of behavior could guide physical simulation was what inspired PhysInk’s SIMULATE button. Still, ASSISTANCE uses arrows instead of direct manipulation for describing movement, which leaves room for ambiguity in the user’s description of behavior. Inevitably, the user resorts to the repetitive process of tweaking and simulation, instead of explicitly demonstrating behavior a single time.

Popović introduced a system to make animation with a physics simulator more usable [16]. Instead of having to define the initial parameters that drive a rigid body simulation, the user can drag objects into desired configurations at specific keyframes. The system then infers the physical parameters that satisfy the user-defined keyframes, and plays back the resulting simulation. In effect, a system similar to ASSISTANCE was created, where the user can sketch structure and interactively guide a simulation of behavior. However, the user must keep defining keyframes until the simulator produces the desired behavior. This is not ideal for users who have a vision of their device’s behavior and who can explicitly demonstrate that behavior, whether or not it is physically-realistic.

Finally, Borchardt introduced the task of causal reconstruction [6], where an internal model is built from a textual description of physical behavior. The model can then be used to answer questions in order to demonstrate an understanding of the input description. Borchardt suggested transitions as a way to model arbitrary physical events, such as chemical reactions caused by light hitting film or steam forming on a metal plate. A transition represents the change in a physical object’s attribute over time (e.g. distance between the steam and plate decreasing as they collide). The input description is reconstructed as a sequence of transitions in a directed acyclic graph, capturing both the meaning of events (changes in physical attributes) and the causal-

ity between events. Although PhysInk's events are less general than Borchartd's, this early AI work was important for arriving at the timeline data structure.

Chapter 6

Discussion

6.1 Future Work

6.1.1 Supporting More Types of Behavior

In future work, features could be added to support a wider range of physical mechanisms. First, movement and contact events are the only phenomena captured by PhysInk. For machines that involve sensors or other abstract stimuli that set a sequence of events into motion, PhysInk can be extended to include arbitrary changes in physical attributes, as is done in Borchardt's work on transitions [6].

Second, PhysInk's demo mode is best suited for linear types of behavior such as those found in Rube Goldberg machines, where a single stimulus leads to a chain of events. This can be restrictive if one considers the fact that many machines are interactive and involve arbitrarily-timed stimuli. For example, the manual transmission of a car cannot be effectively described unless the user is able to demonstrate the effects of changing from one gear to another, and back again. Another example is an autonomous robot which drives back and forth until its infrared sensor receives a signal. Introducing these arbitrarily timed events involves changes to the user interface, as well as the timeline, but would open PhysInk up to an important class of physical behavior.

6.1.2 Three-Dimensional Behavior

Transforming PhysInk into a 3D description tool would be valuable in many cases, for making the user interface feel more natural. The choice to implement PhysInk in 2D was made to simplify the sketch understanding process, allowing us to focus on the problem of capturing behavior. However, if a 3D physics engine could be integrated with a natural user interface for describing 3D structure (for example Google SketchUp [2] or Bae’s ILoveSketch system [5]), then all the same ideas about demonstrating behavior and capturing it in a timeline still apply.

6.1.3 Improving the Search for Physically-Correct Behavior

When the user wants to generate a qualitatively-equivalent physical simulation of their device, PhysInk’s simulator often faces a slow, high-dimensional search through many physical parameters. As devices get more complicated, the parameter space gets larger. Furthermore, there can be a nonlinear relationship between the parameter space and the simulation’s qualitative match distance, which renders standard parameter search methods ineffective (e.g. hill climbing).

An important subject for future work is improving the speed and quality of this search. We imagine doing this by 1) putting parameters into groups that can be searched independently, and 2) performing partial simulation instead of simulation of the entire behavior. This is motivated by the fact that the distance between the simulated and original timelines is calculated in a piecewise fashion. For example, recall the egg cracker’s behavior and in particular, the first ‘leg’ of its behavior: the ball falling and hitting the rotating platform. This first leg can be simulated on its own and compared to the original timeline. Clearly, the mass of the knife is unrelated to this first leg and can be left out of the search, reducing its complexity. Once a solution for the first leg is found, the second leg can be analyzed, and so on until the simulated timeline matches the user’s timeline. This would break up the parameter search into much smaller, independent pieces and would speed up the simulation process significantly.

6.2 Applications

6.2.1 Product Design

A system that can capture natural descriptions of physical behavior has clear applications in product design and engineering. Researchers have found that sketching is an integral step in early stages of product design [11, 14]. Also, Dahan et. al. conducted a study [8] on low-cost methods for testing product concepts in parallel. They found that when using animation to demonstrate product concepts to focus groups, the feedback closely mirrored actual market performance, indicating that animation is an effective medium for communicating physical designs. Despite these observations, pen-and-paper is far more prevalent than software tools in the early stages of product design, mainly due the difficulties of animating physical interactions and the focus of CAD tools on capturing structure rather than behavior. PhysInk can potentially fill this gap by providing an intuitive interface for capturing, editing, archiving and sharing physical behavior.

6.2.2 Video Game Design

We also see applications for PhysInk in the game design industry, particularly in the case of simple 2D mobile phone games. These games are often driven by the Box2D physics engine - the same physics engine integrated into PhysInk. Our system could be used to prototype mechanisms used in the game, or to quickly sketch out new stages or levels for an existing game. Specifically, PhysInk's simulator could help find physical parameters that match the game designer's demonstrations. This would essentially be a what-you-see-is-what-you-get (WYSIWYG) tool for mobile game design.

6.2.3 Programming and Other Domains

Soliciting and capturing descriptions of behavior can lead to useful tools in domains other than the physical world. For example, in programming, students often learn

about data structures and algorithms using sketched diagrams and storyboards. Consider the process of reversing a linked list. The student might sketch the linked list as a number of rectangular nodes, with arrows used to depict pointers between the nodes. The diagram would then be re-drawn in a series of frames to demonstrate the pointer manipulations necessary to reverse the list.

Although the domain is different, this is still a description of behavior - an algorithm's behavior. Accordingly, many of the ideas we have introduced in this thesis can be applied to visual programming: structure can be sketched and interpreted in terms of variables (arbitrary shapes), pointers (arrows) and values (numbers, text or other variables). Algorithms can then be described by manipulating variables, pointers and values. Instead of enforcing physical constraints, the system can enforce programming constraints: variables can be type-checked, and untracked nodes in a linked list can fade away to indicate they are lost to the programmer. Once an understanding of the algorithm has been constructed, it might be possible to produce code and run it on new input values. This is just one example of how an understanding of behavior can lead to useful and intelligent design tools in other domains.

Chapter 7

Conclusion

In this thesis we presented PhysInk, a sketching application that provides a natural way to describe physical structure and behavior to a computer. In sketch mode, the user's strokes are interpreted as physical objects and constraints in a 2D world. The objects become interactive, while constraints are enforced to give the diagram physicality: rigid bodies cannot overlap and joints restrict an object's degrees of freedom. In demo mode, the user can move objects on the canvas to explicitly demonstrate and record the device's behavior. At any time, simple speech commands can be used to label objects and events. Finally, the captured behavior can be played back, exported and shared with others.

We then described the timeline, which is used to store descriptions of behavior. While it records the user's literal manipulations so they can be played back verbatim, the purpose of the timeline is to build a more general, flexible understanding of behavior. For this reason, the timeline also interprets these manipulations as distinct physical events (movement and contact), which are ordered in a graph where edges imply causality. The events capture the qualitative geometry of movement and contact between objects. For example, an object's movement is interpreted relative to other objects (e.g. passing over or under another object). Contact events are viewed in terms of which surfaces collide with each other (e.g. the ball hits the top of the platform).

We showed that capturing behavior in this framework - events, their qualitative

geometry and the causality between them - leads to a more intelligent conversation with the user about function. First, it enables assisted editing, where PhysInk propagates changes in behavior through the timeline in order to maintain a logical sequence of events at all times. Second, it allows users to produce physically-realistic animations of behavior that they have demonstrated a single time. PhysInk achieves this by searching for physical parameters of objects resulting in a simulated timeline that is qualitatively equivalent to the user-described timeline. Determining this equivalence relies on the flexible understanding captured in the timeline's data structures. The implementation of these features demonstrates both the fidelity and utility of timelines.

Finally, we discussed future work for PhysInk and potential applications of capturing behavior in other domains. In summary, by integrating sketch recognition, a physics engine and speech understanding, we created an intuitive interface for describing physical behavior. The timeline used to capture behavior makes PhysInk the first in a new class of design tools that builds and uses an understanding of function, as well as form.

Bibliography

- [1] Cmu sphinx: Open source toolkit for speech recognition. <http://cmusphinx.sourceforge.net/>.
- [2] Google sketchup. <http://sketchup.google.com/>.
- [3] poly2tri: A 2d constrained delaunay triangulation library. <http://code.google.com/p/poly2tri/>.
- [4] Christine Alvarado and Randall Davis. Resolving ambiguities to create a natural computer-based sketching environment. In *ACM SIGGRAPH 2006 Courses, SIGGRAPH '06*, New York, NY, USA, 2006. ACM.
- [5] Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. Ilovesketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st annual ACM symposium on User interface software and technology, UIST '08*, pages 151–160, New York, NY, USA, 2008. ACM.
- [6] Gary C. Borhardt. Understanding causal descriptions of physical systems. In *Proceedings of the tenth national conference on Artificial intelligence, AAAI'92*, pages 2–8. AAAI Press, 1992.
- [7] Erin Catto. Box2d: A 2d physics engine for games. <http://box2d.org/>.
- [8] Ely Dahan and V. Srinivasan. The predictive power of internet-based product concept testing using visual depiction and animation. *Journal of Product Innovation Management*, 17(2):99–109, 2000.
- [9] Richard C. Davis, Brien Colwell, and James A. Landay. K-sketch: a 'kinetic' sketch pad for novice animators. In *Proceedings of the twenty-sixth annual SIGCHI conference on Human factors in computing systems, CHI '08*, pages 413–422, New York, NY, USA, 2008. ACM.
- [10] V. Domiter and B. Zalik. Sweep-line algorithm for constrained delaunay triangulation. *Int. J. Geogr. Inf. Sci.*, 22(4):449–462, January 2008.
- [11] D.L. Jenkins and R.R. Martin. Importance of free-hand sketching in conceptual design: Automatic sketch input. *ASME*, 1993.

- [12] Dan Klein. The stanford parser: A statistical parser. <http://www-nlp.stanford.edu/software/lex-parser.shtml>.
- [13] Dan Klein and Christopher D. Manning. Fast exact inference with a factored model for natural language parsing. In *In Advances in Neural Information Processing Systems 15 (NIPS)*, pages 3–10. MIT Press, 2003.
- [14] B. Macomber and M. C. Yang. The role of sketch finish and style in user responses to early stage design concepts. In *Proceedings of the 2011 ASME Intl Design Engineering Technical Conferences*, 2011.
- [15] Michael Oltmans and Randall Davis. Naturally conveyed explanations of device behavior. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.
- [16] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 209–217, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.