Escape Machine: teaching computational thinking with a tangible state machine game

Michael Philetus Weller
CoDe Lab
Carnegie Mellon University
Pittsburgh, USA
philetus@cmu.edu

Ellen Yi-Luen Do
ACME Lab
Georgia Tech
Atlanta,USA
ellendo@cc.gatech.edu

Mark D Gross
CoDe Lab
Carnegie Mellon University
Pittsburgh, USA
mdgross@cmu.edu

ABSTRACT
We present a methodology for building objects-to-think-computationally-with and illustrate its application in developing our Escape Machine game. The input mechanism for this game is a tangible state machine built with Posey, our computationally enhanced construction kit. Through manipulating this state machine children create an algorithmic specification for the behavior of both the avatar and its enemies in an attempt to navigate a maze without being eaten. We outline several strategies for success at Escape Machine and discuss how it embeds an important computational thinking concept in interaction with a tangible device.

ACM Classification Keywords
K.3.1 [**Computers and Education**]: Computer Uses in Education; H.1.2 [**Models and Principles**]: User/Machine Systems; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces, Input devices and strategies

INTRODUCTION
Frank Lloyd Wright credited his aptitude for design to early childhood experiences with wooden blocks and other educational "gifts" designed by German educator Friedrich Froebel [3]. Wooden blocks and other similar toys can exert such a powerful influence on children because they provide more than narrow skill training; these toys provide an entire framework for children to understand the world around them, and an intoxicating glimpse of the satisfaction of discovery. Papert and others have championed the project of enhancing these *objects-to-think-with* [13, 18] with computation to provide an even more fertile playground for exploration. The work presented here is part of our effort to support what Wing calls "computational thinking" [22] by creating *objects-to-think-computationally-with*.

The motivating insight for teaching children computational thinking is that many concepts that computer scientists apply to manage complex coding tasks such as iteration and decomposition into parts can also usefully be applied more generally to understanding the world around us. Through early education in core computational thinking we can provide all children with the cognitive tools to think about complex phenomena, while providing the next generation of computer scientists with a strong foundation for further education. As the field is still in its infancy, one of the most pressing projects is enumerating a set of core computational thinking concepts as well as methods for presenting them. Therefore we have developed a methodology for creating objects-to-think-computationally-with that engage children in a particular computational thinking concept. We

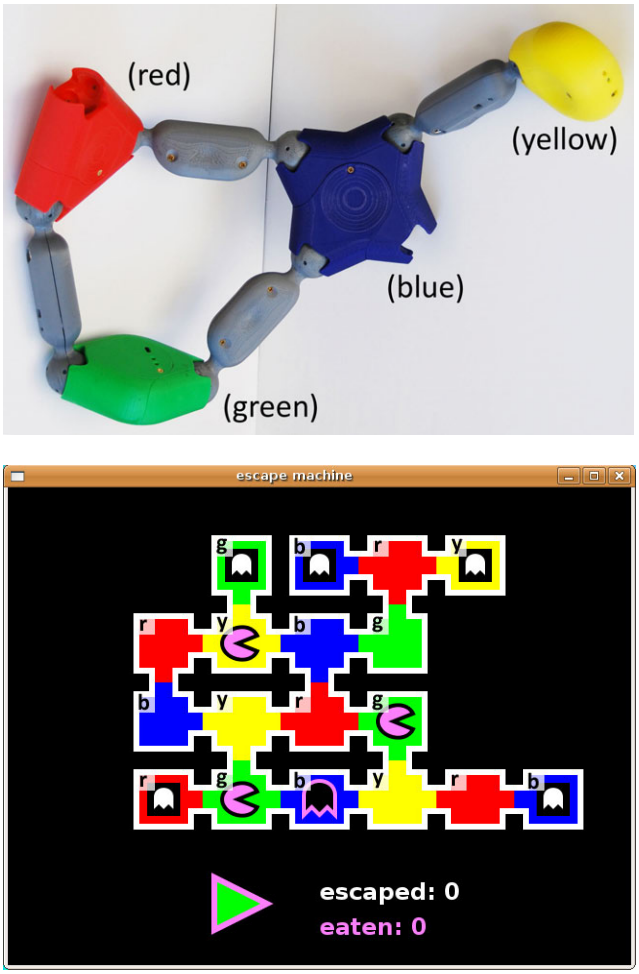

Figure 1. Top: state machine built with Posey kit; bottom: screen image of Escape Machine maze.

IDC '08, June 11-13, 2008 Chicago, IL, USA

illustrate our methodology by describing how we have applied it in building our Escape Machine game.

Escape Machine is a puzzle game. It is controlled with a tangible state machine built with a few parts of Posey, a hub-and-strut construction kit instrumented to capture the configuration and orientation of constructions its users build [21]. The goal of our Escape Machine application is to provide a platform for play with a basic state machine, a powerful abstraction that is central to computer science. By constructing and reconstructing the tangible state machine you control the movements of your avatar, the black ghost, through a maze (Figure 1). Your goal is to help all your fellow ghosts imprisoned in the maze escape. However the state machine you build also controls the pink ghost eaters who are out to eat both the imprisoned ghosts and you. The challenge is to build state machine configurations that lead you towards the imprisoned ghosts while holding the ghost eaters at bay. We have tried to design game play so that discovering successful strategies demands understanding the basic mechanisms of specifying behaviors algorithmically in terms of a state machine.

In the following section we describe our methodology for building objects-to-think-computationally-with using Escape Machine as an example. Next we discuss related work. Then we explain the rules of Escape Machine and walk through one level of game play. We give examples of successful game play strategies to illustrate how applying these strategies requires children to comprehend how a state machine specifies a system's behavior. Finally we discuss the implications of our methodology and directions for future work.

METHODOLOGY

We have developed a methodology generalized from work on Logo [8, 13] for the design of objects-to-think-computationally-with to support a particular core computational thinking concept. Our goal is to construct environments where children are encouraged to exercise these concepts to manipulate the behavior of the system. Such a system would:

1. Promote mastery of a core computational thinking concept.
2. Encourage exercising the concept as a tool for manipulating behaviors.
3. Maintain interest through engaging interaction.

Logo is an excellent example of a system that fulfills these desiderata. Although which concepts should be on the list of core computational thinking concepts is still an open question, several concepts presented by Logo such as variables, functions, and iteration are excellent candidates.

Logo also arguably fulfills the second desideratum with respect to these concepts. For example it is possible to construct all sorts of behaviors for the turtle without using iteration (or recursion), but it quickly becomes tiresome to type so much, to the point that to have any fun you really need to figure out iteration (or recursion). Mastery of the concept is central to exercising control over the turtle.

Merely playing with systems that are based on computational concepts does not necessarily convey the concept: consider the game of tic-tac-toe. Writing a program to play tic-tac-toe is an excellent introductory exercise for students to explore the concept of a game tree, as the space of possible games is relatively small. However, merely *playing* tic-tac-toe is not especially conducive to discovering the idea of a game tree, as most people intuitively grasp the winning strategy after only a few games and are not particularly motivated to think about abstractions that could help them achieve further mastery.

As for our third desideratum, Logo's widespread dissemination is a tribute to its ability to maintain children's interest. The interpreter provides immediate gratification, requiring only a few lines of code to generate interesting turtle behaviors, but is powerful enough to tackle non-trivial computer science challenges. However Logo's strength is also its limitation: it is a thinly disguised full-strength Lisp development environment. For the project of promoting computational thinking to a broad audience, we hope to develop some stepping stones along the path to becoming a Logo hacker.

Computationally Enhanced Objects-to-Think-Computationally-With

A key aspect of our effort to build more accessible objects-to-think-computationally-with is the adoption of computationally enhanced construction kits [4, 5] as a platform. These kits build on the strengths of toys such as Froebel's Blocks [3] and Tinkertoy [http://www.hasbro.com/playskool/tinkertoy/] by embedding computation, sensing and actuation to imbue systems with additional behaviors and feedback. For example Topobo [15] and roBlocks [19] both allow children to build creatures and imbue them with behaviors purely through physical interaction. Anderson et al.'s self-describing blocks [2], Triangles [9], Senspectra [11] and our Posey kit [21] are construction kits that can be manipulated to interact with applications on a host computer. These systems illustrate the particularly engaging and accessible interaction computationally enhanced construction kits afford.

Application of Methodology in Escape Machine

Our high level goal in developing the Escape Machine game has been to demonstrate the potential of computationally enhanced construction kits as a platform for situated exploration of computational thinking concepts. Two factors influenced our decision to address the concept of finite state machines: our Posey kit is well suited to modeling graph structures; and our experience teaching undergraduate architecture students to design systems with embedded computation has made it painfully clear that many students are poorly prepared for specifying behaviors algorithmically. By providing a tangible state machine as

an input mechanism to specify the behavior of animated characters in a game we hope to illustrate the potential for even relatively simple state machines to generate complex behavior.

To create an engaging environment we designed a puzzle game around our tangible state machine input device. By challenging players with the goal of helping the ghost rescue its friends without getting eaten we hope to motivate them to master the state machine interface. We designed the rules of play to encourage players to consider the relationship between the general specification of preferences with the tangible state machine and the resulting actions that the game characters will take. We tuned game play to be complex enough to require players to think through the implications of different possible state machine configurations, but not so complex as to be inaccessible.

BACKGROUND AND RELATED WORK

Objects-to-think-with

This work builds on the educational tradition of learning through engaging self-guided activities rather than lecturing and memorization. One foundation of this tradition is the object-to-think-with, an artifact intentionally crafted to support discovery and learning.

Froebel [3] introduced a series of what he called *gifts* for young children that supported a variety of activities incorporated into the first kindergarten curriculum. One of the most famous gifts was a set of modular wooden blocks for constructing buildings and other shapes similar to those found on the shelves of a toy store today.

There are a wide variety of commercial construction kits that have not been the subject of scholarly research but have served as inspiration. Tinkertoys are an early hub-and-strut kit with a mix of static and rotationally free connectors. Zoob is a more recent entry into the hub-and-strut genre with poseable ball and socket connectors. Other systems such as Lego, Erector and Meccano provide complex construction systems capable of realizing a wide variety of functional mechanical systems.

Feurzeig and Papert [8] created Logo to leverage the power of computing to create an environment for children to explore and discover computational concepts. Early versions of Logo drove a physical robot turtle around on a piece of paper [13], later versions were entirely screen-and-keyboard based. Feurzeig and Papert were among the first to explicitly identify computation as a promising medium for extending the tradition of objects-to-think-with.

Computationally Enhanced Construction Kits

Resnick and others at the MIT Media Lab have advanced the approach of using physical media to express and explore computational ideas [12, 17, 18]. Eisenberg and Wrensch applied these techniques to construction kits [4, 23]. These projects range from systems to describe 3D geometry to kits of parts to assemble functional robots. All provide tangible interfaces for interacting with computational systems.

Our Posey kit (Figure 1, top) [21] is a computationally enhanced poseable hub and strut construction kit. Its hub and strut form maps to model anything that a graph structure can describe, for example an articulated skeleton, a chemical molecule, a kinematic linkage or a building structure. Posey employs a ball and socket connection that allows users to move the parts of an assembled model. Hubs and struts are optocoupled through the ball and socket joints using infrared LEDs and photosensors. Wireless transmitters in the hubs send connection and geometry information to a host computer. The host computer assembles a representation of the physical model as the user creates and configures it. Application programs can then use this representation to control domain specific computational models.

Several early efforts provided tangible interfaces for describing 3D geometry. Aish's three dimensional Building Block System [1] enabled architects to input models to a CAD system. Frazer et al.'s 3D input devices [7] enabled designers to build models that interface with software that can give design advice. "Self-describing blocks" [2] facilitate computer modeling with instrumented Lego-style snap-together plastic blocks.

In Triangles [9], a construction kit of flat plastic triangles interfaced to a computer, each tile corresponds to a different application, such as an email client or a personal calendar; and in a later version, to a character or object in a story. Mechanical and electronic magnetic connectors support constructing a variety of geometric forms that correspond to a suite of applications.

Glume [14] is composed of soft modules each with six stubby arms filled with hair gel that communicate conductively to determine the overall model topology. Glume presents an interesting and original tactile quality, but geometry is not directly sensed, only inferred from topology. Although the form of these various kits differ, in each a user can make only static and rigid configurations.

Monkey [6] is a specialized input device for virtual body animation. It is basically an artist's lay figure instrumented for use as an input device. Instead of constructing a simulation of human animation and locomotion using a screen interface, the animator poses and moves the Monkey to define the character's animation. However, although Monkey demonstrates the potential of this form of tangible interaction, it cannot be reconfigured and can only be used to control one particular (humanoid) geometry.

Schweikardt and Gross's roBlocks [19] are cubes that connect with magnetic data connections and provide either sensing, actuation or logic functions. By assembling a group of roBlocks you simultaneously define the system's physical form and interactive behavior.

Speech-Enabled Alphabet Blocks [5] send data packets over an infrared optocoupled serial connection to build up words from the letters on the top face of the blocks and then speak the word out loud.

Topobo [15] is a construction kit of articulating vertebra-like pieces for building models with embedded kinetic memory. A Topobo construction is composed of a few active hubs with sensing, communications and actuation, and a variety of passive limb components that attach to the hubs and each other to construct a model of a creature. Each hub records angular movement at its joints when a button is pressed, and then replays the same movement in a loop with its motors after the button is released. Users build a creature, move the model across a terrain, and then watch the model replay its movement from its embedded kinetic memory. The Backpacks project [16] extends Topobo by adding timers and sensors that modulate the reproduction of recorded movements. These projects demonstrate how engaging a kinematic construction kit interface can be, but neither Topobo nor its Backpacks have sensors to detect the topology or geometry of most of the pieces in the kit.

Senspectra [11] is a flexible hub-and-strut construction kit. It is composed of balls with several embedded female headphone jacks that serve as hubs and flexible struts with male headphone jacks at each end. The struts bend to allow connecting hubs and struts in a wide variety of configurations, but are not poseable—sensors in each strut detect the degree, but not the direction, of bending. Although Senspectra's connectors are under-sensed, in models with many hubs and struts the constraints imposed by the topology allow Senspectra to infer the model's overall geometry.

Tetrobot [10] is composed of hubs and actuated struts that can be combined to form actuated truss structures. Støy's Odin system [20] also presents a kit of parts for assembling actuated truss structures. It is composed of hubs and struts that connect to form space-frame structures as well as an electrical bus to transmit power and control signals. Some struts have ball and socket joints at one or both ends, and others contain linear actuators that extend and contract the strut. By strategically combining different sorts of struts, subassemblies capable of taking steps or other actuations can be created.

PARC's Polybot chain-type module [24] is capable of self-reconfiguration. Each module has two hermaphroditic connection plates with a rotational actuator in between. A series of modules can be connected together to form a snake or loop. By adding another passive cubic module with six connection plates, configurations with legs can also be made. Assemblies of these modules can perform rolling, snaking, and walking gaits.

Computational Puzzle Games

There is a rich tradition of puzzle games that are not necessarily electronic but support computational thinking. We were inspired by many of these games to adopt turn-taking puzzle-solving game play for Escape Machine.

Think-A-Dot [http://www.geocities.com/jaapsch/puzzles/thinkadot.htm] is a toy with mechanical logic flip-flops arranged to display a pattern on the front of the case. The player drops a marble in one of three slots at the top of the toy and several gates both direct the path of the marble and change state. The challenge is to create certain patterns by dropping marbles into the slots in the sequence.

Dr. Nim [http://www.cs.rit.edu/~ark/museum/drnim01.shtml] is a mechanical toy that allows a human to play against the mechanism at the game of Nim, the object of which to avoid taking the last marble.

Chipwits [http://chipwits.com] is a computer game developed for the original macintosh computer in which players use an iconic programming language to program on-screen robots to navigate on-screen mazes.

RoboRally [http://www.roborally.com] is a board game that allows players to specify programs for their robots by selecting five cards from those dealt to them and placing them face down. Each round of play then unfolds in stages as all players each turn over cards in unison, and move their game pieces around the board accordingly. Players whose program has not wrecked their robot are dealt a new hand and select a new five-card program.

These last two games share a key feature with Escape Machine: the player, through some mechanism, programs the behavior of game characters. In Chipwits the programming language is icon based; in RoboRally the program is expressed in cards, and executed manually by the players; and in Escape Machine the players program the behavior in the game by manipulating a physical model that expresses a finite state machine.

GAME DESIGN

Several gaming traditions influence our Escape Machine. We have adopted the iconography and gestalt of Pac-Man [http://www.mameworld.net/pacman/] which introduced a simple yet engaging mode of game play: steering a character to clear a series of markers from a maze without being eaten by monsters. However, our game-play mechanism eschews Pac-Man's raw test of reflexes and instead relies on puzzle-solving. Escape Machine is a game of strategy.

We were also influenced by more recent games that abandon the joystick in favor of specialized input devices such as Guitar Hero's [http://www.guitarhero.com/] guitar controller. Guitar Hero is not using a game to teach guitar, but rather leveraging the guitar as an interaction mechanism to promote engaging play. With Escape Machine's tangible state machine input we aim to leverage finite state machines to fashion an interaction mechanism that is both fun to play and satisfying to master.

Goals

Our design for Escape Machine has several goals. Foremost is to create a game play mechanism based on the algorithmic specification of behavior through state machines. This mechanism should not just be *amenable* to thinking in terms of state machines, it should *promote* thinking in terms of state machines as successful strategy.

Another goal for Escape Machine is to demonstrate the use of our Posey construction kit to support accessible and engaging application interfaces through sensing the player's manipulations of the model's shape and orientation.

Our final goal for Escape Machine is to create a game that is fun to play. Our puzzle-style maze game inverts the roles of the Pac-Man characters: the player's avatar is a ghost, and the goal is to prevent the ghost-eaters from eating you and your fellow ghosts. The tangible state machine game controller brings the control mechanism into the physical space of the room. We mean to encourage experimentation and collaboration as strategies to achieve the satisfaction of creating desired behaviors and solving the puzzles.

Rules of Play

Escape Machine is played on a maze composed of a connected set of red, green, blue or yellow rooms as shown in the screen snapshot in Figure 2 (right). The Escape Machine provides a maze in which no room connects to more than three others and no two of a room's neighbors have the same color. The colors of the rooms in the maze correspond to the colors of the four hubs in the controller that the player holds and manipulates to program the behavior of the characters in the maze. Figure 2-left shows a diagram of the tangible state machine controller (see also Figure 1-top). It is essential to note that the different colored hubs also have different numbers of connecting sockets—yellow has one; green has two; red, three; and blue has four connections.

A maze contains three kinds of characters: the ghost (in the middle, red, room in Figure 2) is the player's avatar in the game; imprisoned ghosts (like the one in the yellow room at the left) wait helplessly to be either rescued by the ghost or eaten by a ghost eater (the Pac-Man figure in the green room at the right). Your goal is to guide the ghost to rescue as many imprisoned ghosts as you can before they are eaten by ghost eaters, while avoiding being eaten yourself.

Leaf nodes of each maze are initially populated with imprisoned ghosts. When the ghost moves into a room with an imprisoned ghost, it helps that ghost escape and you receive a credit. If a ghost eater enters a room with an imprisoned ghost, it eats that ghost and you receive a demerit. If a ghost eater enters the same room as the ghost you are eaten and the game is over. When all imprisoned ghosts have escaped or been eaten, the maze is clear and you move on to a new maze.

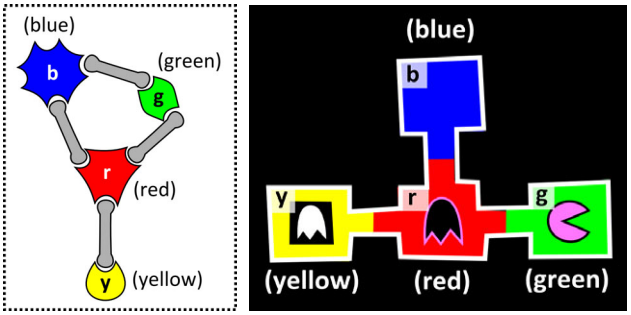

Figure 2. Diagram of state machine (left) and maze fragment (right).

Play proceeds in a sequence of discrete steps. On each step the player reconfigures and/or re-orients the tangible state machine to program the characters' actions. Then the characters move according to that state machine configuration. Ghost eaters move first, then the ghost.

The configuration of the tangible state machine (Figure 2-left) controls the movement of both the ghost and ghost eaters. It specifies which room a character will move to in the next turn. For example, in Figure 2-left, the connections in the state machine graph specify that a character in a blue room can move only to a neighboring red or green room. If, as in this case, the state machine graph allows more than one move (i.e., red or green) then the character's move depends on which hub is (physically) higher or lower. The ghost prefers the lower hubs (in this case red), while the ghost eaters prefer higher hubs (in this case green). Thus the specification of the character's behavior is sensitive to the topology, position and orientation of the tangible state machine.

To determine where the ghost in the red room (in Figure 2-right) will move in the next turn we first look at the state machine. We see that the red state connects to all three other color states (blue, yellow, and green). Then we look at the colors of neighboring rooms in the maze. The red room is also connected to rooms of all three colors. The state machine graph specifies more than one possible move, so each character chooses based on the vertical order of the state hubs. The ghost prefers the lowest one (yellow in Figure 2), so it will move into the yellow room on the next turn. The yellow room in the maze contains an imprisoned ghost, so when the ghost enters that room it will enable the imprisoned ghost to escape.

Unfortunately, before the ghost may move all the ghost eaters must move. To determine where the ghost eater in the green room will move we look at the tangible state machine again. Green connects to both blue and red, so both moves are allowed, and blue is preferred because it is higher. In this case, however, the room the ghost eater is in connects only to one room (the red room) so there is only one choice. The ghost eater will move into the red room and eat us before we have a chance to move!

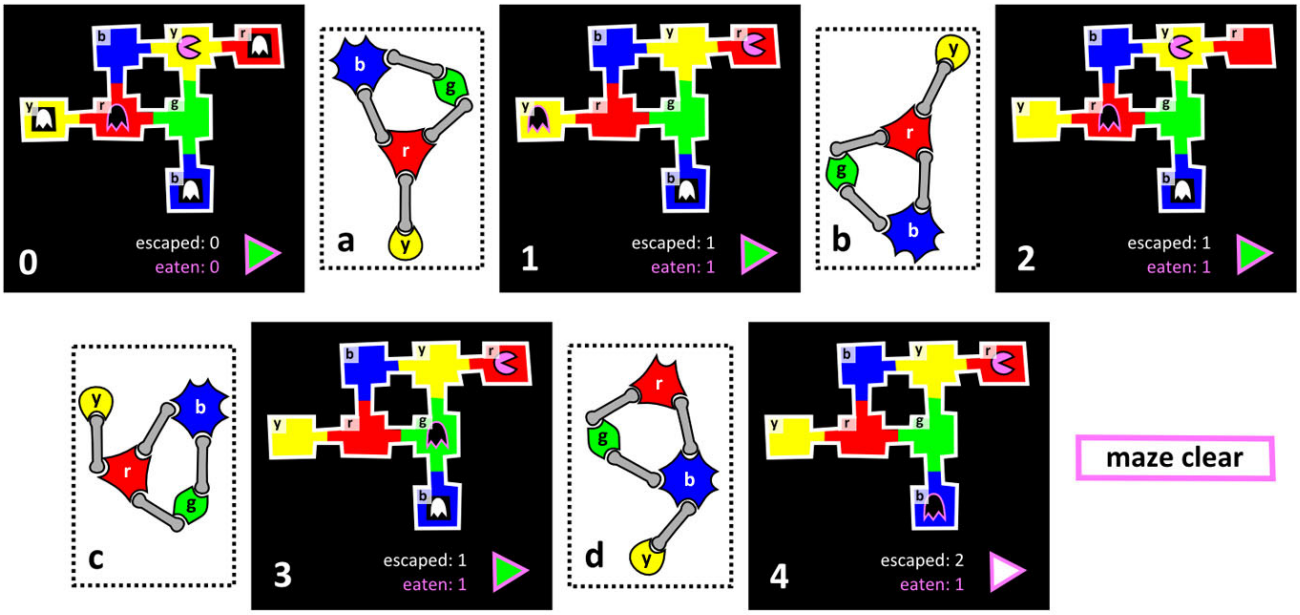

Figure 3 Sequence illustrating one round of play for a simple maze.

Example Game Play

The sequence in Figure 3 shows the movements of the ghost and a ghost eater through a simple maze according to a series of state machine configurations. Screen *0* shows the maze. The state machine is initially arranged as shown in *a*, yellow hub down. The ghost eater is in a yellow room, and in the state machine yellow connects only to red, so when the player presses Go, the ghost eater moves to the adjacent red room and eats the ghost imprisoned there (oops).

The ghost is in a red room with adjacent rooms of each color, and the red hub is connected to the hubs of all three colors, so (as explained above) the ghost moves to the color of the lowest hub, yellow. The ghost imprisoned there escapes. Screen *1* shows the state after all characters have moved.

For the second move, the player has not reconfigured the tangible state machine but just *turned it upside down*, (shown in *b*) causing both the ghost and ghost eater to return to their original positions (screen *2)*.

For the third move the player *bends* the tangible state machine (*c*), placing the green hub below the yellow and blue hubs. The ghost will drop to the green room as shown in screen *3*; but the ghost eater can still only move back into the adjacent red room. For the final state machine configuration in *d*, the player *disconnects* the yellow hub from the red hub and *reattaches* it to the blue hub, trapping the ghost eater in the red room. The whole assembly is then rotated so that the ghost slides down from the green room to the blue room and rescues the last imprisoned ghost (screen *4*).

STRATEGIES FOR SUCCESSFUL PLAY

Although the maze shown here is small and somewhat trivial to complete, it illustrates the fundamental tension of Escape Machine: the physical constraints of the state machine make the colors of the rooms important. Yellow rooms work as buffers as it is fairly easy to manipulate which other (one) color the yellow hub attaches to, whereas the blue hub is always attached to at least two other colors. Care must be taken to move the ghost in the desired direction without letting the ghost eaters run amok.

We have attempted to make game play accessible and engaging, while encouraging children to think abstractly about specifying behavior algorithmically. At each turn the movement of the ghost and ghost eaters provides feedback for children to validate their expectations of behavior expressed by the current game controller construction. We have tuned game play to allow children to discover the following strategies for success, and others, through experimentation.

Imprison Ghost Eaters

This strategy is demonstrated in the game play example above. If a player is willing to sacrifice the ghost imprisoned in a leaf node then a ghost eater can often be trapped in that node by disconnecting (in the state machine) the color of that leaf node from that of its neighbor in the maze. In steps 3 and 4 above, the ghost eater is trapped in a red leaf node room; its only way out is through a yellow room, but in the state machine red does not connect to yellow. If at some point it becomes necessary to release the imprisoned ghost eater (by reconnecting red and yellow in the state machine) for the ghost to make progress, the ghost

eater will still remain trapped cycling between the two nodes. By exercising this strategy children develop an understanding of how connections between nodes in the state machine limit the motion of characters through the maze.

Isolate Portions of the Maze with Valves

Some mazes are divided into two subgraphs by a single room. When the room is yellow or green a player can use it as a one-way valve to contain ghost eaters in one area of the maze while the black ghost travels safely around the now ghost-eater-free area rescuing imprisoned ghosts. By exercising this strategy children develop the ability to reason about extended patterns of behavior through multiple turns.

Identify Valley Paths

Because the ghost prefers (physically) lower state machine nodes and ghost eaters prefer higher nodes, in certain situations one can construct a state machine configuration that creates peaks and ridges for the ghost eaters to occupy, leaving a safe valley pathway past them. Exercising this strategy helps children to identify higher-order patterns that they can leverage to generate desirable behaviors.

DISCUSSION

It is difficult in a paper to convey a sense of what it is like to play what seems like an intricately complicated game. (In fact, the game is far easier to understand by playing it than reading about it might suggest). One might well ask, "But is it fun?" Although we have not yet engaged children in user studies, a dozen or so students at our university have played the first working versions of Escape Machine. The game proceeds quite slowly, at the pace of chess, as players ponder the consequences that their state machine construction will have on the characters' actions in the maze. In informal testing, it was not unusual for a player to take ten or fifteen minutes for each move. What we found remarkable is that, despite this slow pace of play, people remained strongly engaged in the game! We also observed that people enjoyed playing together in pairs, thinking aloud as they contemplated their next move. In informal discussions players reported that the game was intriguing, unusual, and fun. We plan to conduct more formal studies involving players of different ages.

We plan to assess the Escape Machine with respect to the goals we set out above. Can people play it, and is it engaging and fun? Our preliminary testing shows that people can and it is, but we would expand the evaluation to include a wider demographic. Does it promote mastery of the core concept of a state machine? In a sense, if they learn to play the game successfully they have mastered the core concept, although this does not guarantee that they can transfer what they have learned to other situations. We could include play with Escape Machine in a college level computer science course to see whether it helps students understand state machines more easily. Our main purpose, though, is not to help CS students, but to provide children a foundation for computational thinking. To assess whether Escape Machine serves that goal, we might learn a lot simply by listening to the conversation among Escape Machine play partners as they play through the game. Do they acquire or develop language for talking about the game that reflects a structural understanding that in turn might map to other domains?

More formally we might design an evaluation that requires using state machine concepts to solve a problem in a different (non-game) domain, and see whether people who have mastered Escape Machine can apply what they have learned to the new domain. And of course there are a set of studies to perform that compare tangible interaction in Posey with a screen-based state machine diagram to control the game.

The Escape Machine tangible state machine game controller illustrates our methodology for creating objects-to-think-computationally-with. The system engages players in tangibly manipulating simple algorithmic specifications of behavior. Although the abstractions involved are simple, we believe that they provide an accessible taste of the power of algorithmic specification, and hope that they will whet children's appetites to tackle more powerful environments such as Logo.

Clearly, Escape Machine is not—and is not intended as—an open-ended constructionist environment like Logo. Compared with Logo, Escape Machine is a very limited microworld: it does not have variables or functions, conditionals, or iteration, and all you can do is control the actions of some simple game characters.

Yet Escape Machine shares one essential property with Logo and other programming environments: the player controls behavior in a microworld by constructing a symbolic representation. In the case of Logo, the symbolic representation is a program expressed in code. In Escape Machine it is the connections, geometry, and orientation of the Posey model, which represents a finite state machine graph. Whereas a finite state machine is a powerful computational abstraction usually reserved for college computer science students, Escape Machine conveys this concept directly through manipulating a physical model. To program the state machine requires writing no code; only connecting, reconnecting, and reorienting the Posey model.

Escape Machine's tangible interface—made up of Posey construction kit parts—functions quite differently from a traditional joystick controller or keyboard game interface. Whereas keyboard (e.g., arrow key) controls and joysticks provide instantaneous input to drive game characters around the screen (up, down, left, right, fire!) in the Escape Machine players engage in constructing a *description* that drives game play. That is, conventional game interfaces employ direct manipulation; Escape Machine on the other hand quite deliberately demands indirect description. Players are, in a simple but essential sense, programming.

This last observation—that by configuring and manipulating the pieces of a construction kit, players are programming—points to a relatively unexplored and potentially promising area of application for computationally enhanced construction kits: physical programming. Our other applications of Posey—for example chemistry and puppeteering—capture and map physical manipulations to a domain modeled in the computer. But Escape Machine's domain is itself computation: and the physical constructions that players make are themselves representations of computational behavior. The particular affordances of the Posey kit and mapping these to a state machine are less important than the general idea that construction kits can provide powerful interfaces to computation. They then become not merely computationally enhanced construction kits, but physical construction kits for computation.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their helpful comments. This work was supported by the National Science Foundation under ITR-0326054.

REFERENCES

1. Aish, R. 3d Input for CAAD Systems. *Computer-Aided Design*, *11* (2). 66–70.
2. Anderson, D., Frankel, J., Marks, J., Leigh, D., Ryall, K., Sullivan, E. and Yedidia, J. Building Virtual Structures with Physical Blocks *User Interface Software and Technology (UIST)*, ACM, 1999, 71-72.
3. Brosterman, N. *Inventing Kindergarten*. H. N. Abrams, New York, 1997.
4. Eisenberg, M., Buechley, L. and Elumeze, N. Computation and Construction Kits: Toward the Next Generation of Tangible Building Media for Children *Cognition and Exploratory Learning in the Digital Age (CELDA 2004)*, Lisbon, Portugal, 2004, 423-426.
5. Eisenberg, M., Eisenberg, A., Gross, M.D., Kaowthumrong, K., Lee, N. and Lovet, W. Computationally-enhanced Construction Kits for Children: Prototype and Principles *Intl. Conf. of the Learning Sciences (ICLS)*, Seattle, USA, 2002, 79–85.
6. Esposito, C., Paley, W.B. and Ong, J. Of Mice and Monkeys: A specialized input device for virtual body animation *Symposium on Interactive 3D Graphics (I3D)*, ACM, 1995, 109–114.
7. Frazer, J., Frazer, J. and Frazer, P. New Developments in Intelligent Modelling. *Computer Graphics*, *81*. 139–154.
8. Fuerzeig, W., Papert, S., Bloom, M., Grant, S. and Solomon, C. Programming-Languages as a Conceptual Framework for Teaching Mathematics. *ACM SIGCUE Outlook*, *4* (2). 13-17.
9. Gorbet, M.G., Orth, M. and Ishii, H. Triangles: tangible interface for manipulation and exploration of digital information topography *Human Factors in Computing (CHI)*, ACM, Los Angeles, USA, 1998, 49-56
10. Hamlin, G.J. and Sanderson, A.C. Tetrobot: a modular approach to parallel robotics. *Robotics & Automation*, *4* (1). 42-50.
11. LeClerc, V., Parkes, A. and Ishii, H. Senspectra: A computationally augmented physical modeling toolkit for sensing and visualization of structural strain *Human Factors in Computing (CHI)*, ACM, San Jose, USA, 2007, 801–804.
12. McNerney, T.S. From turtles to Tangible Programming Bricks: explorations in physical language design. *Personal Ubiquitous Computing*, *8* (5). 326-337.
13. Papert, S. *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc., New York, 1980.
14. Parkes, A., LeClerc, V. and Ishii, H. Glume: exploring materiality in a soft augmented modular modeling system *Human Factors in Computing (CHI)*, ACM, 2006, 1211–1216.
15. Raffle, H., Parkes, A. and Ishii, H. Topobo: A constructive assembly system with kinetic memory *Human Factors in Computing (CHI)*, ACM, 2004, 647-654.
16. Raffle, H., Parkes, A., Ishii, H. and Lifton, J. Beyond record and play: backpacks: tangible modulators for kinetic behavior *Human Factors in Computing (CHI)*, ACM Press, 2006, 681–690.
17. Resnick, M., Bruckman, A. and Martin, F. Pianos not stereos: creating computational construction kits. *interactions*, *3* (5). 40-50.
18. Resnick, M., Martin, F., Berg, R., Borovoy, R., Colella, V., Kramer, K. and Silverman, B. Digital manipulatives: new toys to think with *Human Factors in Computing (CHI)*, ACM Press, Los Angeles, USA, 1998, 281-287.
19. Schweikardt, E. and Gross, M.D. roBlocks: a robotic construction kit for mathematics and science education *Intl. Conf. on Multimodal Interfaces (ICMI)*, ACM, Banff, Canada, 2006, 72-75.
20. Stoy, K., Lyder, A., Garcia, R.F.M. and Christensen, D. Hierarchical Robots *Workshop on Self-Reconfiguring Robots at Intelligent Robots and Systems (IROS)*, IEEE, San Diego, USA, 2007.
21. Weller, M.P., Do, E.Y.-L. and Gross, M.D. Posey: Instrumenting a Poseable Hub and Strut Construction Toy *Tangible and Embedded Interaction (TEI)*, ACM, Bonn, Germany, 2008, 39-46.
22. Wing, J.M. Computational Thinking. *Communications of the ACM*, *49* (3). 33-35.
23. Wrensch, T. and Eisenberg, M. The programmable hinge: toward computationally enhanced crafts *User Interface Software and Technology (UIST)*, ACM, San Francisco, USA, 1998, 89-96.
24. Yim, M., Duff, D. and Roufas, K. PolyBot: A Modular Reconfigurable Robot *Intl. Conf. on Robotics and Automation (ICRA)*, IEEE, 2000, 515-519.