MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

A Framework for Microworld-style Construction Kits

Carol Strohecker, Adrienne H. slaughter

TR2000-19 December 2000

Abstract

We describe a genre of game-like construction kits and an extensible Java framework that models method and structures for generating them. We include descriptions of the kitsćonceptual bases and design principles, and explain how work with users of the K̈it for Kitsf̈ramework informed modifications promoting use by people with a broad range of expertise in programming, design, multimedia production, and learning theory. Continued development of the framework is informed by museum-based contexts for kit use.

201 Broadway, Cambridge, Massachusetts 02139

MITSUBISHI ELECTRIC RESEARCH LABORATORY

A Framework for Microworld-style Construction Kits

Carol Strohecker
Adrienne Slaughter

TR2000-19 May 2000

Abstract

We describe a genre of game-like construction kits and an extensible Java framework that models method and structures for generating them. We include descriptions of the kits' conceptual bases and design principles, and explain how work with users of the "Kit for Kits" framework informed modifications promoting use by people with a broad range of expertise in programming, design, multimedia production, and learning theory. Continued development of the framework is informed by museum-based contexts for kit use.

A Framework for Microworld-style Construction

Carol Strohecker
MERL - Mitsubishi Electric Research Lab
201 Broadway
Cambridge, MA 02139 USA
+1 617 621 7517
stro@merl.com

Adrienne Slaughter
Stanford University
c/o 310 W. Michigan Avenue
Clinton, MI 49236 USA
+1 517 456 4822
ahs@alum.mit.edu

ABSTRACT

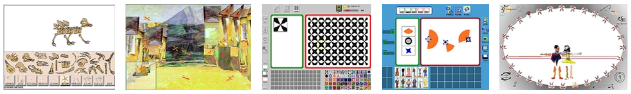
We describe a genre of game-like construction kits and an extensible Java framework that models method and structures for generating them. We include descriptions of the kits' conceptual bases and design principles, and explain how work with users of the "Kit for Kits" framework informed modifications promoting use by people with a broad range of expertise in programming, design, multimedia production, and learning theory. Continued development of the framework is informed by museum-based contexts for kit use.

Keywords

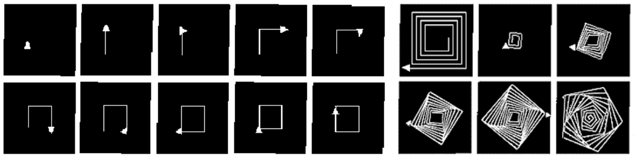
Java framework, construction kit, learning, interaction design

INTRODUCTION

We are developing a Java framework to facilitate implementation of a genre of kits based on principles of Constructionist learning [16, 17, 18, 24, 25, 46]. The kits bear a family resemblance to Tinker Toys™, LEGOs™, and other building toys, but are dynamic and especially visual. Currently the kits are prototyped as highly interactive 2D graphical software, but we project iterative development including augmentation with tangible input and output devices, and concurrent development of social contexts supporting the playful building activity.



The kits are inspired by the notion of "microworlds," perhaps best exemplified by Logo-based Turtle Geometry [1, 24]. This computational world elegantly models basics of differential geometry: the graphical "turtle" is characterized by just two properties, position and heading. Users (typically children) interact by typing commands to effect operations of movement. The turtle responds by translocating forward, turning, and so on, leaving a trace as it goes. Delightful pictures result, which can become complex quickly. Thus children play with ideas in building-block fashion as the properties of a vector ground development of further understandings – of angles, the geometries of squares and spirals, etc.


[1]

Some of our kits are intended for children, and others are intended for people of all ages, but all involve similar modeling of a conceptual domain. Players build with basic elements and operations, and then activate the constructions. Dinosaur skeletons balance as they walk and run; maps reveal street-level views; geometric tiles form symmetric patterns; animistic creatures spawn, maintain, and disrupt social distances; and dancers' breathing patterns determine cyclic timing for a shared dance.

Various collaborators have developed versions of these kits during the past several years. Commonalities of design and implementation, as well as conceptual underpinnings, have led us to formulate a "Kit for Kits" (K4K), extensions to Java's Abstract Windowing Toolkit (AWT) that facilitate development of the growing genre. Because the kits are so strongly visual, much of the Kit4Kits supports creation of structure, function, and appearances of objects. K4K works with Java for capturing and dealing with actions and elaborates Java2D facilities for transforms, shapes, painting, and image-handling to provide highly customizable, manipulable objects.

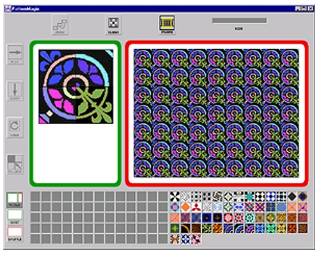
Four existing construction kits and designs for a fifth informed development of a first version of K4K. We made it available to people who used it for developing their own kits. These experiences helped us to improve the framework. While implementing the refinements, we continued using K4K to develop our own new kit. Next steps include creating facilities for sound and for web-supported play.

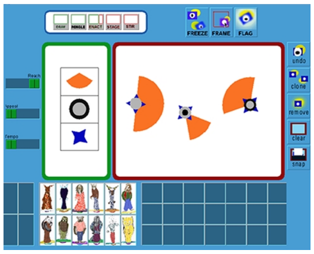
K4K BEGINNINGS

PatternMagix players build colorful tiles and spread them into mosaic-like patterns (below left). Panels change size as the player moves from the build area to the activation area, modeling the turn-taking of a dialog [2, 5, 7, 47, 48]. PatternMagix was the third of our microworld-style kits but the first to be implemented in Java (version 1.1.6). The straightforward implementation involves rectangular building elements and filters effecting geometric

[1] These illustrations are adapted from [24, 72-73].

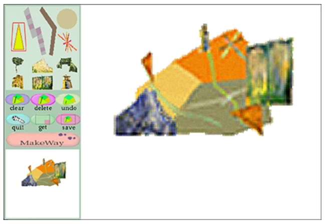
transformations of the tiles. We added image buttons and structures for looping series of image buttons as animations.

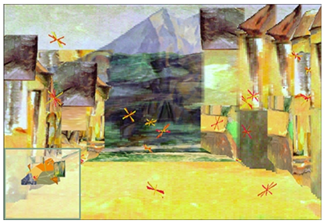




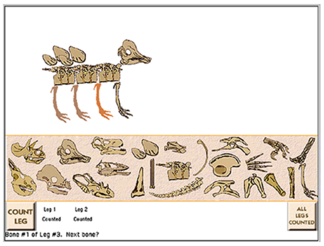
AnimMagix players select and adjust behavioral attributes that simulate social behaviors of animistic creatures (above right). Varying degrees of awareness and attraction affect the creatures' movements as they interact with one another [2, 5, 7, 47, 48]. The AnimMagix interaction design is again based on the idea of a conversation between the player and the system. Its implementation (in Java 1.1.8) makes use of many of the same structures as PatternMagix. We added filters to achieve color blending as creatures' perceptual fields overlap, and structures enabling transparent backgrounds for the image files containing creatures' animistic costumes.

WayMaker players diagram a city, real or imagined, by forming a map from geographic primitives such as districts, paths, and landmarks (below). The software then transforms scale, view, and representation to illustrate a stroll through the mapped environment. The dynamic illustration takes the form of a frame-by-frame animation that preserves topological relationships among the primitive structures [21, 22, 34, 36, 37, 38, 39]. The current version (in Java 1.1.8) is the second WayMaker prototype. Building elements are Java polygons, as in AnimMagix, but we added facilities for manipulations like scaling and stretching. We also elaborated the construction process by enabling specification of particular images, such as towers and mountains for landmarks, which appear in both the maps and path views. A miniature echoes the building of the map and maintains its screen position as the display changes to reveal street-level views. Variable transparency filters allow visibility of views beneath the miniature map and effect edge blending between image segments. We also developed processes and structures for saving and retrieving constructions so they can be extended and/or re-viewed.

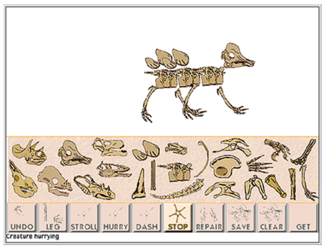




Bones players put dinosaur parts together and then test the skeletons to see if they can balance while standing or moving (below). Based on a constructed creature's number of legs, the location and mass of its center, and a user-selected speed of movement, the software analyzes structural integrity and whether the creature can maintain balance as it moves. Animations, including a rich set of gait patterns, derive from the literature on biomechanics and animal locomotion [3, 19, 27, 32]. The current version (in Java 1.1.8) is the third Bones prototype. As in PatternMagix, the building elements are image files, but we added a requirement for arbitrarily shaped polygonal images rather than rectangular ones. This change had profound implications for image production, for mouse event structures and manipulability, and for filters that effect geometric transformations and size and color changes. We also developed a two-phase construction process in which players first assemble a creature and then indicate which bones are meant to constitute the legs. This process shifted a burden from the software to the player but complicates the interaction design. As in WayMaker, Bones players can save constructions as images and retrieve them as objects that can be reactivated and manipulated. This capability will become the starting point for web-based trading of constructions.



Bone #1 of Leg #3. Next bone?



Creature hurrying

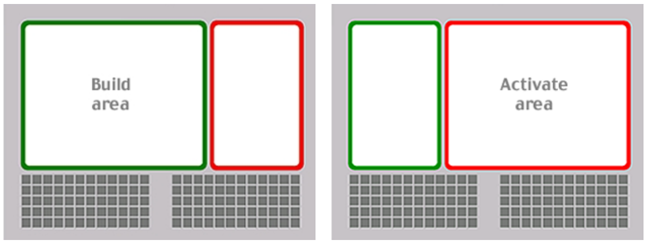
"Microworld" approach

The design of these kits relates to the "microworld" approach exemplified by Turtle Geometry, in which fundamental elements and operations characterize a conceptual domain. A program-assigned mass value and a player-determined position characterize the elements in Bones. When combined to locate the overall center of mass for a composited creature, these properties determine whether it will maintain its structural integrity and balance. The conceptual domain is motion study; more particularly, it is balance; still more particularly, it is the role of center of mass in balancing. Focusing on the most salient features of a conceptual domain is one of our design challenges, as is developing manipulable representations that people can build with. For WayMaker, the domain is topology, basic relations of proximity. The building elements are representations of districts, edges, paths, nodes, and landmarks; operations consist of positioning the elements with respect to one another. For PatternMagix, the domain is geometric symmetry. The building elements are square-shaped tiles, and the operations are rotation and reflections. For AnimMagix, the domain is social dynamics. The elements are representations of sensori-motor/attract functions, and the operations consist of adjustments to degrees of perceptivity, sociability, and motility.

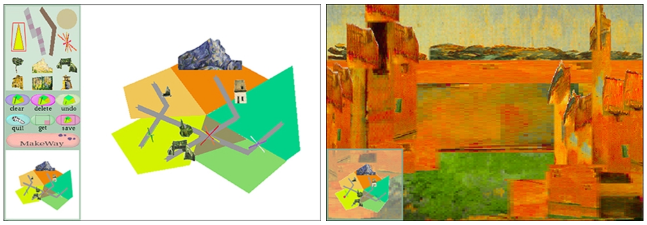
Design highlights

Consistent with our "screen-first" approach, we have formulated design heuristics of "object permanence," "transparency," and "multiple simultaneous views" [c.f. 16, 29]. We recommend designing miniature representations or other recall mechanisms for screen areas, buttons, and other devices, so that when they are not in use they are still present and easily accessible [43]. Visualizations of algorithms, calculations, and processes are the biggest design and implementation challenges but are crucial to this highly visual genre [42, 44]. We represent constituent properties of objects, often in ways that facilitate users' modifications of them, and we provide various forms of visual and aural feedback so that results of actions are apparent. Comparisons help people to perceive the shifts of scale, perspective, time, and representation that can be fundamental to understanding dynamic phenomena. Through the use of graphic treatments such as side-by-side views, miniature displays, and the like, we address the principle that you understand something best when you understand it in more than one way [23]. Kits employing such treatments may also be more accessible to a range of users with diverse thinking styles [45].

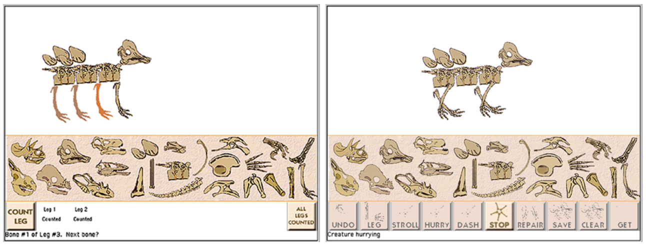
Adjustments of panel sizes in Magix exemplify one kind of visualization. The changes provide concrete representation of the distributed control between the between the player and the system as their "conversation" proceeds.



Side-by-side comparisons of the miniature map next to path-level views in WayMaker provide shifts of scale, perspective, and representation to facilitate players' notice of relative placements of elements.



Bones includes a diagrammatic visualization of the center-of-mass calculation and projection of the center to the base formed by the creature's points of contact with the ground. This is a representation of static balancing. Leg movements act as visualizations of gait patterns. In the next version we hope to improve the visualizations of dynamic balancing and to add footprints as complementary visualizations of gait patterns.



Bone #1 of Leg #3. Next bone?

Creature hurrying

K4K DEVELOPMENT

The Kit4Kits framework includes the "elements and operations" conceptual basis as well as Java structures acknowledging the importance of visual presentation to the genre of kits [16, 24, 35, 40]. We chose Java for object orientation in implementing the highly interactive systems, and in anticipation of cross-platform, web-friendly requirements as communities of players evolve.

During the course of the work, key aspects of our requirements and of Java were changing. Java 1.1.8 introduced a new event model; with Java2 came Java2D, which provided for complex screen areas, new image objects, transformations, custom cursors, and other helpful functionality. However, Java2D greatly slowed down image painting and introduced scores of bugs. We considered Swing, newly incorporated into the Java2 JDK, but it was still too nascent for our tastes and more extensive than our needs. Some of the bugs were resolved in Java 1.3 beta, in which K4K is currently implemented. Meanwhile we had increasingly complicated wants from Java. In the beginning, it was acceptable to use a rectangle to describe the bounds of a particular object; soon it had to be a polygon. Originally we wanted to rotate a square by increments of 90 degrees; soon we wanted to rotate complex polygons by 3 degrees and have corresponding images also rotate so they could fill the polygons exactly. Originally transparent backgrounds in image files were sufficient; soon we needed transparency filters for image blending as well as a range of filters beyond geometric transformations, such as color changes and gradients. The basic classes evolved as well.

Version 1

The first version of K4K includes several kinds of classes: screen areas, building elements, widgets, and structures for managing them. Everything happens within a MainFrame, or top-level window. The Mode layout manager determines what objects need to be in the MainFrame when, and enables specification of their absolute positions on the screen.

Elements are components that implement mouse events, specific geometric forms, and transformations. ImageElements are Elements that implement image loading and painting. As players build with Elements they form Constructions, similar but more complex structures useful in activation processes.

Java Panels are not flexible enough for our notions of building and activation, so we created Zones, containers that can take on various shapes and implement special painting code such as borders and background images. A Composer is a kit-specific Zone that holds Elements and allows them to be manipulated within it. An Arena is another kit-specific Zone that holds Constructions and performs the necessary functionality for them to interact with each other or otherwise activate. A Bench is the third kit-specific Zone, which displays snapshots of saved Constructions, allowing them to be accessed individually or from a file.

Java's standard button functionality was not rich enough for the state changes and visual signalling that we needed, so we developed particular kinds of buttons. An ImageButton uses an image for painting. A SpeedButton continues to execute when the mouse is pressed and held on it: the longer the hold, the faster the button performs its action. An AnimButton performs an animation of a sequence of images when the mouse rolls over the button. A SpinButton displays a number that can be modified through some action like pressing an up or down arrow.

The OutputPanel is a feedback mechanism that typically appears at the bottom of the screen and displays textual messages. A ToolTip provides for a short, descriptive string to appear over a component in the interface.

Commands are objects that encapsulate instructions for responding to particular events. Commands remember all aspects of what they do, primarily so these actions can be undone. StateObserver uses Commands to keep track of undoable and redoable events.

TimeContext keeps track of time and notifies TimeListeners registered with it that some specified period of time has elapsed. Many TimeListeners can be notified at once. An object implementing the Selectable interface can respond to a mouse event by being selected. This has implications for some associated action, such as the object painting itself differently. Selectable objects register themselves with SelectListeners.

Two interfaces enable special pictorial treatments. Trailable specifies that a component will leave some kind of depiction in its wake, such as a trail of fading images. Traceable specifies that a component will leave a trace as it moves, such as a line describing its path.

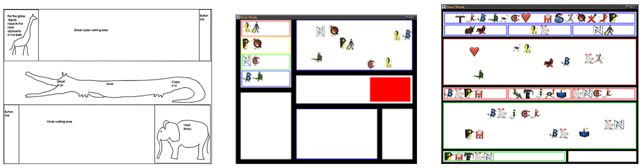
Usage trials

To test the usefulness and robustness of these facilities, we invited people interested in producing "computer games more meaningful than shoot 'em ups" to try out the K4K. An interesting mix of people responded. One participant was fluent in Java but had little visual or interaction design experience; another had a lot of non-Java programming experience and extensive knowledge of computer games and their interaction design; another participant had some experience with Java and as an architect had extensive visual design knowledge; still another had no programming experience but expert knowledge in linguistics and cognition. This diversity was fortuitous: because of our application-orientation we had been focusing on players of the kits as "users," but from the perspective of K4K they are end-users. We needed now to address ranges and extents of expertise for people who would want to use a framework like K4K.

We organized the workshop as a design studio. During the course of a few days we hoped to explain the microworlds approach and design principles, and to work with participants in using K4K to develop cursory examples of construction kits. Demos and supporting documentation described our existing kits, their conceptual grounding, design, relevant digital image production, K4K documentation, and coding examples showing how K4K could be applied to create each of our kits. There was insufficient time to do a thorough "elements and operations" analysis for the kits that participants proposed. We addressed these concerns but emphasized a "screen-first" approach, which brought designers and developers together by centering implementation around devices on-screen. The approach also helped us to clarify the genre's "build – activate – save – trade" premises, emphasizing focuses on end-users' constructions of graphical objects and potential behaviors, the software's activation of constructions, and ultimately, social contexts in which saving and trading the constructions can enhance learning.

One participant wanted to make a kit with which players could explore timing relationships in the context of music-making. A ball falling on a soundpad makes a sound, which can be specified as a particular tone. Building consists of adding ball and soundpad elements to the Composer. Upon activation, each ball strikes its soundpad, and the Arena displays strike patterns in a graph-like notation resembling a musical score. The patterns could be saved for replay. In this design, as in Bones, the build and activate processes share a screen area but constitute distinct activities. Thus our Composer and Arena constructs were suitable.

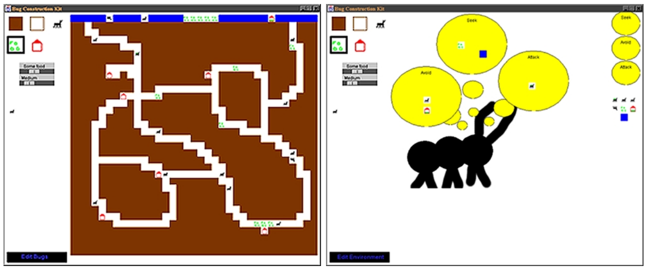
Two other participants pooled their skills in programming, visual design, and linguistics. They began a kit with which players can build words (below). Their design evolved through several arrangements of screen areas and corresponding work flow:[2] Eventually they settled on an arrangement based on downward movement as the player progresses through a process of word building: letters combine to form phonemes, which become syllables that form words. Letters must match according to particular sonority rules in order to form a phoneme [20]. Matches are saved into pockets ordered according to position of the phoneme within a word: an onset phoneme combines with a vowel to begin a word, which ends with a coda syllable. Saved words may or may not yet appear in an English dictionary, but must follow the onset-vowel-coda pattern. The collaborators carefully separated the screen areas according to each of these functions, yet the main areas support both building and a kind of activating, which takes the form of checking for proper letter matches and syllable patterns. They implemented both areas by extending our Composer structure, rather than using the Composer for one and the Arena for the other. Composers typically handle operations on elements, which in this case take the form of validity checking.



Another participant wanted to make a simulation kit that would deal with notions of ecology. He wanted players to be able to control aspects of the environment, which

[2] Graphical letterforms are from [30].

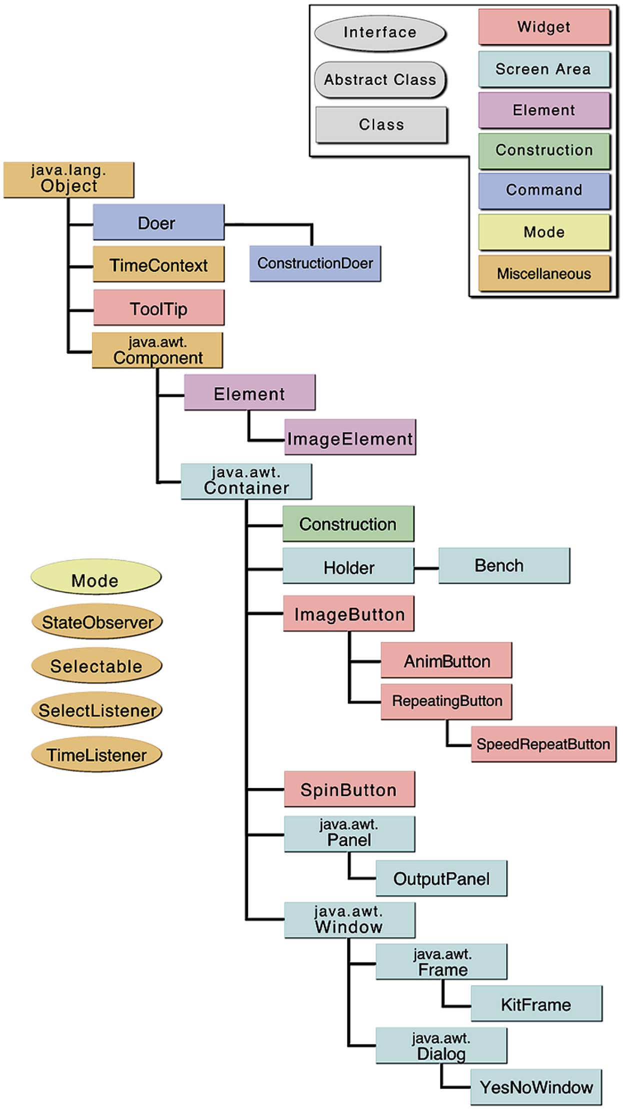
resembles an ant farm, and creatures that inhabit it, which he called "bugs" (below). He separated the two functions into screen displays that differed but maintained common features. Both modes include both build and activate processes. In environment mode, the player can add bugs and food for the bugs while the simulation is running. In bug mode, the player can specify rules governing bugs' properties and behaviors, such as being hungry, seeking or avoiding food, seeking or avoiding other bugs, seeking food stores, dying when hungry and not finding food, and so on. At first he represented the rule structure as a kind of logical chart, but through discussion moved to more graphical representations. This notion of build mode is like the specification phase of building in our prototypes. He implemented this specification functionality by extending our Composer structure.



Version 2

In addition to general debugging, we made structural modifications to K4K based on these experiences. We concluded that maintaining the notions of build and activate was worthwhile, but these functions could happen in many different ways. Our separate Composer and Arena constructs therefore seemed overspecified. In the next version of K4K we collapsed them as a single structure, the Holder, which maintained the flexibility inherent in the Zone structure but had implications for some of the image treatments and control mechanisms. We also rethought our notion of Mode, which had implicitly incorporated three concerns: what the player is doing at a given time, what is happening on the screen at a given time, and what a Construction is doing at a given time. We concluded that it would be most helpful to separate these concerns and distribute them among the relevant Holder, Construction, and Command structures.

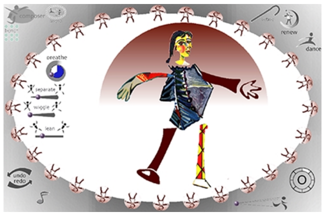
We added functionality for new classes: CommandDoer, RepeatingButton, KitSlider, ImageSlider, and YesNoWindow. We also developed methods for necessary but potentially cumbersome considerations like double buffering and loading images and sounds. We protected and included setters and accessers for all variables, and ensured that instances of an object could not be created without initializing relevant variables. Additionally, we reduced the number of abstract classes, improved the code for several classes, and refined the comments overall. We changed some terminology so it would pertain more specifically to our effort: MainFrame became KitFrame and Command became Doer. We also changed the names of some methods so they were more compatible with AWT terminology. Generally we aimed to make all the structures as accessible and malleable as possible.

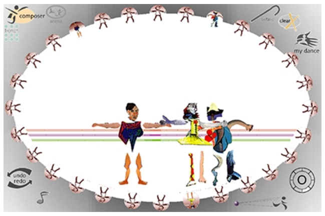


Many of the modifications were geared toward enhancing ease of use for people with less Java programming experience than we had originally envisioned. We have not attained a "low threshold, high ceiling" profile for K4K as an environment in which to learn programming, but that could become an interesting direction for the work. Nevertheless, in addition to Java code and documentation, the K4K includes code examples for specific kits and documents addressing varying degrees of expertise in programming, visual design, interaction design, multimedia production, and learning theory. We include code-starter "seeds" for creating screen areas, Elements, Constructions, widgits, and Doers, and for basic data and control structures. The documents also answer simple questions that first-time K4K users would typically ask, and briefly describe the Java interface model. For more experienced Java users, K4K provides the Java API and relevant documentation.

Application

Our fifth kit, Zyklodeon (implemented in Java 1.3 beta), helped initially in identifying functionality for K4K and subsequently in testing the code for bugs, completeness, and extraneous constructs. Zyklodeon is the most complex of the kits we have implemented, mainly because of the shape and arrangement of items on the screen (below). As in Bones, the building elements are arbitrarily shaped polygonal image files, which in Zyklodeon represent a dancer's arms, legs, torso, and head. Players build by composing dance figures and setting properties that affect their movements and cyclic timing for a shared dance. We added arbitrarily shaped image buttons, a two-tiered structure for saving and retrieving constructions, and a greatly elaborated system of event-triggered visualizations. In this world of time/space relationships, dancers' movements visualize breathing cycles, progress through a shared dance cycle, and characteristics affecting choreographic concerns such as leap moments and heights [4, 10, 15, 26, 32, 41].





RELATED WORK

During the course of this work we compared our concerns and output to other software construction kits, authoring tools for simulations and multimedia, environments for learning about programming basics, and Java frameworks. There is a large amount of interesting work in each of these related areas. While the distinguishing characteristic of our effort remains the basis in microworld theory, we would hope to develop or maintain features that several notable projects have also achieved or pointed toward, should our kits and the Kit4Kits move beyond prototype to more widely disseminable forms.

Other software construction kits, such as SimLife, SimCity, Incredible Machine (and its sequel, The Even More Incredible Machine), Lemmings, Tom Snyder Productions, ToonTalk, and HyperGami, resemble our kits in various ways. Several of these could be described by some aspects of the "build – activate – save – trade" mantra. Some include a range of media as we would ultimately like for our kits. Tom Snyder Productions, for example, distributes some kits with printed booklets and activities, and HyperGami includes user-designed printed output in the construction process [8].

It is interesting that authoring tools for simulations – such as SimLife, SimCity, Microworlds Pro, AgentSheets / Visual AgenTalk [28], Cocoa, and Stagecast (KidSim) – tend to have more in common with environments for learning about programming basics than do multimedia authoring tools such as 3D MovieMaker, Macromedia Director, and NACDRAW (though of course a tool like Microworlds Pro can be used for both). Although it is still not clear to what extent we hope K4K will become an environment for learning about programming basics, we have learned from environments such as Microworlds Pro, ToonTalk, Cocoa, AgentSheets / Visual AgenTalk, Squeak, SmallTalk [13, 14].[3]

The two Java frameworks we have found most beneficial to consider are the Interactive Illustrations produced at Brown University and the growing movement toward developing HCI patterns. We believe that despite the Constructionist grounding of the Interactive Illustrations and their similarity to our genre of kits, both the kits and K4K maintain a stronger focus on users' and end-users' deliberations and constructions [31]. We considered presenting K4K as a kind of pattern language for microworld-style construction kits, but feared engendering a "cookie-cutter" approach that would work against flexibility [9, 11, 12, 42]. We feel that including various forms of support for novice programmers will prove a more fruitful direction.

K4K FUTURES

In the immediate term we aim to develop better facilities for handling sound and for facilitating kit players' web-based trading of constructions. We are looking into recent versions of Swing and Java Beans to see whether they may now be more robust and compatible with our effort. Up to now Beans have not sufficiently addressed reliance on imagery to the extent that our genre demands, but we anticipate evolution of Beans and their potential compatibility with our notion of "seeds." We also plan improvements to our existing kit prototypes, which will contribute to further K4K facilities. Toward this end we have begun collaboration with exhibit and program developers at Boston's Museum of Science. Several of the kits are currently installed there as test exhibits that visitors can try and critique. In addition to improving the kits and K4K, these trials will help in developing museum-based social contexts for long-term kit use. We anticipate developing separate versions of each kit, to support episodic use within exhibit areas and extended use within studio-like environments in museums, homes, and other situations that ideally would be networked to one another. Including tangible input and output for the kits will help to effect craft-based activities with learning communities.

ACKNOWLEDGMENTS

Bones was developed between 1993 and 2000 by Carol Strohecker, William Abernathy, Dan Gilman, John Shiple, and AARCO medical illustrators. WayMaker was developed between 1996 and 2000 by Carol Strohecker, Barbara Barros, Adrienne Slaughter, Dan Gilman, and Maribeth Back. PatternMagix was developed between 1996 and 1997 by Edith Ackermann, Carol Strohecker, and Aseem Agarwala. AnimMagix was developed between 1997 and 1998 by Edith Ackermann, Carol Strohecker, Aseem Agarwala, Adrienne Slaughter, and Dan Gilman. Zyklodeon was developed between 1999 and 2000 by Carol Strohecker, Adrienne Slaughter, Mike Horvath, Noah Appleton, Nadir Ait-Laoussine. The Kit4Kits was developed between 1999 and 2000 by Adrienne Slaughter

[3] See www.lcsi.com, www.toontalk.com, www.crim.ca/~hayne/Cocoa/, www.agentsheets.com, www.squeak.org.

and Carol Strohecker. In addition to visitors to Boston's Museum of Science, Nadir Ait-Laoussine, Cara Brooks, Kristoff Brooks, Larry Burks, Jutiki Twan Gunter, Surapong Lertsithichai, Pau Sola-Morales, and Mark Strehlow have participated in usage trials for various kits. John Evans, Aradhana Goel, Tim Gorton, and Milena Vegnaduzzo participated in usage trials for the Kit4Kits. Mike Eisenberg, Doug Smith, Tom Wrensch, Glenn Blauvelt, and Matthew Brand have participated in development of models for museum-based learning environments. We also thank David Cavallo, Adele Goldberg, Stephanie Houde, Sarah Kuhn, Fred Martin, Seymour Papert, Barry Perlman, Vicki Porter, Vennila Ramalingam, Warren Sack, Tim Shea, Brian Silverman, and Doug Smith for helpful discussions during the course of this work. The effort is supported by Mitsubishi Electric Research Laboratory.

REFERENCES

1. Abelson, H., and DiSessa, A. *Turtle Geometry: The Computer as a Medium for Exploring Mathematics.* MIT Press, Cambridge, MA, 1980.

2. Ackermann, E., and Strohecker, C. Build, launch, convene: Sketches for constructive-dialogic play kits. MERL TR99-30, Mitsubishi Electric Research Laboratory, Cambridge, MA, 1999.

3. Alexander, R. M. *Dynamics of Dinosaurs and Other Extinct Giants.* Columbia Univ. Press, New York, 1989.

4. Armitage, M. (ed). *Martha Graham: The Early Years.* Da Capo Press, 1978.

5. Bakhtin, M. *The Dialogic Imagination.* Univ. of Texas Press, 1981.

6. Berger, J. The moment of Cubism, in *The Look of Things,* ed. Stangos. Viking Press, 1974.

7. Bruner, J. *Actual Minds, Possible Worlds.* Harvard Univ. Press, Cambridge, MA, 1986.

8. Eisenberg, M. and Nishioka, A. Creating polyhedral models by computer. *Journal of Computers of Mathematics and Science Teaching* (1997).

9. Erikson, T., and Thomas, J. Putting it all together: Pattern languages for interaction design. Proceedings of CHI'97, 226. See also the summary report of the workshop at http://www.pliant.org/personal/Tom_Erickson/Patterns.WrkShpRep.html.

10. Freedman, R. *Martha Graham: A Dancer's Life.* Clarion Books, 1998.

11. Gabriel, R. P. The failure of pattern languages. *Journal of Object-Oriented Programming* (1994).

12. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1977.

13. Goldberg, A. What should we learn? What should we teach? *Proceedings of the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications* (1995), 30-45.

14. Goldberg, A., and Robson, D. *Smalltalk-80: The Language.* Addison-Wesley, Reading, MA, 1989.

15. Graham, M. *The Notebooks of Martha Graham.* Harcourt Brace Jovanovich, 1973.

16. Gruber, H. E., & Vonèche, J. J. (eds.). *The Essential Piaget.* Basic Books, New York, 1977.

17. Harel, I., & Papert, S. (eds.). *Constructionism.* Ablex, Norwood, NJ, 1991.

18. Kafai , Y., and Resnick, M. (eds.) *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World.* Lawrence Erlbaum, Mahwah, NJ, 1996.

19. Hildebrand. How animals run. *Scientific American* (May, 1960) 148-157.

20. Kenstowicz, M. *Phonology in Generative Grammar.* Blackwell, Cambridge, MA, 1994.

21. Lynch, K. *The Image of the City.* MIT Press, Cambridge, MA, 1960, 1992.

22. Lynch, K. Reconsidering the image of the city, in *Cities of the Mind: Images and Themes of the City in the Social Sciences,* ed. Rodwin, L. and Hollister, R, M. Plenum Press, New York, 1984, 151-161.

23. Minsky, M. *The Society of Mind.* Simon and Schuster, New York, 1986.

24. Papert, S. *Mindstorms: Children, Computers, and Powerful Ideas.Basic Books,* New York, 1980.

25. Papert et al., http://el.www.media.mit.edu/.

26. Poggi, C. Frames of reference: Table and tableau in Picasso's collages and constructions, in *In Defiance of Painting: Cubism, Futurism, and the Invention of Collage.* Yale Univ. Press, New Haven, CT, 1992.

27. Raibert, M. *Legged Robots That Balance.* MIT Press, Cambridge, MA, 1986.

28. Repenning, A. AgentSheets®: An interactive simulation environment with end-user programmable agents, *Proceedings of Interaction 2000* (Tokyo, Japan).

29. Resnick, M., Berg, R., and Eisenberg, M. Beyond black boxes: Bringing transparency and aesthetics back to scientific investigation. *Journal of the Learning Sciences* (1999).

30. Rey, H. A. *Curious George Learns the Alphabet.* Houghton Mifflin, Boston, 1963, 1991.

31. Simpson, R. M., Spalter, A. M., van Dam, A. Exploratories: An educational strategy for the 21st century. Brown University online ID number: schoolhouse_1449, 1999.

32. Stein, G. *Picasso.* Dover, New York, 1938, 1984.

33. Strohecker, C. A model for museum outreach based on shared interactive spaces. *Multimedia Computing and Museums: Selected Papers from the Third*

International Conference on Hypermedia and Interactivity in Museums, (Archives & Museum Informatics, Pittsburgh, 1995), 57-66.

34. Strohecker, C. Cognitive zoom: From object to path and back again, in *Spatial Cognition II,* Springer Verlag, 2000.
35. Strohecker, C. Construction kits as learning environments. *Proceedings of IEEE International Conference on Multimedia Computing and Systems 2* (Florence, 1999), 1030-1031.
36. Strohecker, C. Toward a developmental image of the city: Design through visual, spatial, and mathematical reasoning. *Proceedings of Visual and Spatial Reasoning in Design* (Massachusetts Institute of Technology, 1999), University of Sydney, 33-50.
37. Strohecker, C. What would Cézanne think? *Proceedings of Creativity and Cognition* (Loughborough University, 1999).
38. Strohecker, C. and Barros, B. WayMaker. *Extended Abstracts, CHI'97* (Atlanta, GA, 1997), ACM Press.
39. Strohecker, C., and Barros, B. Make way for WayMaker. *Presence: Teleoperators and Virtual Environments* 9:1, 97-107, 2000.
40. Strohecker, C., and Slaughter, A. Kits for learning and a kit for kitmaking. *Extended Abstracts, CHI'2000* (The Hague, Netherlands, 2000), ACM Press.
41. Strohecker, C., Slaughter, A., and Horvath, M., and Appleton, N. Zyklodeon. Mitsubishi Electric Research Laboratory, Cambridge, MA, forthcoming.
42. Tidwell, J. Common ground: a pattern language for human-computer interface design. http://www.mit.edu/~jtidwell/ui_patterns_essay.html, 1999.
43. Tufte, E. R. *The Visual Display of Quantitative Information.* Graphics Press, Cheshire, CT, 1983.
44. Tufte, E. R. *Envisioning Information.* Graphics Press, Cheshire, CT, 1990.
45. Tufte, E. R. *Visual Explanations: Images and Quantities, Evidence and Narrative.* Graphics Press, Cheshire, CT, 1997.
46. Turkle, S., and S. Papert. Epistemological pluralism: Styles and voices within the computer culture. *Signs* 16:1, 1990, Chicago University Press, 128-33.
47. Vygotsky, L. *Mind in Society.* Harvard Univ. Press, Cambridge MA, 1978.
48. Wertsch, J. V. *Voices of the Mind.* Harvard Univ. Press, Cambridge MA, 1991.