Idioms for Composing Games with EToys

Markus Gaelli, Oscar Nierstrasz
Software Composition Group
University of Bern, Switzerland
{gaelli,oscar}@iam.unibe.ch

Serge Stinckwich
GREYC, Université de Caen/CNRS, France
serge.stinckwich@info.unicaen.fr

Abstract

[1] *Creating one's own games has been the main motivation for many people to learn programming. But the barrier to learn a general purpose programming language is very high, especially if some positive results can only be expected after having manually written more than 100 lines of code. With this paper we first motivate potential users by showing that one can create classic board- and arcade games like Lights Out, TicTacToe, or Pacman within the playful and constructivist visual learning environment EToys dragging together only a few lines of code. Then we present recurring idioms which helped to develop these games.*

1. Introduction

> Any problem in computer science can be solved with another level of indirection. But that usually will create another problem. — David Wheeler [20]

> Software people tend to favor the joy of complexity, yet we should strive for the joy of simplicity. — Alan Kay

Squeak is a feature-rich, platform-independent open source implementation of the Smalltalk programming language written in itself and by many developers. It includes network- and 2D/3D-graphics support, an integrated development environment, and a constructivist learning environment for children called EToys[9]. EToys are used as a teaching vehicle around the world, including high-schools in the United States, Japan, Spain, France and Germany.

EToys introduce a new way of programming to a wide audience: Traditional, widespread programming languages like Java and C++ are text based and require a strong separation between GUI and underlying model. In contrast, EToys, which are modeled after Self [18], provide an interface where the developer must drag and drop together code which directly manipulates visual objects. By *not* separating the GUI from the model, EToys open up the opportunity to explore a rather different application development paradigm, albeit within a rather constrained environment. We present several fairly complex applications built within EToys and we identify several recurring idioms[2] which allow the developer to keep the GUI and the model together in one object.

By setting up examples of classic games built within EToys and publishing recurring idioms used while developing these games, we hope to motivate and help potential users of Squeak to play around with "this instrument whose music is ideas".

The games presented in this paper as well as many others can be downloaded together with their scripts from the first author's web site [6]. The rest of the paper is structured as followed: First we introduce classic games built with EToys, and then we explain visual idioms used building this games.

2. Games

As space is rather limited we will not explain in detail how we have developed these games with EToys [2]. Instead we will focus on describing their features, showing motivating screenshots, and explaining and categorizing recurring idioms of the solutions.

2.1 14-15 Puzzle

The 14-15 puzzle [7] was invented by Sam Loyd in the 1870s. In the original version the stones 14 and 15 are interchanged, and the (impossible) task for the player is to bring

[1] Fourth International Conference on Creating, Connecting and Collaborating through Computing (C5'06) pages 222–231

[2] We draw the usual distinction between *idioms* and *design patterns*: idioms are implementation techniques closely tied to a particular programming language or programming paradigm, whereas design patterns are recurring micro-architectures that are relatively language-independent. The idioms we present lie somewhere at the boundary between idioms and design patterns.

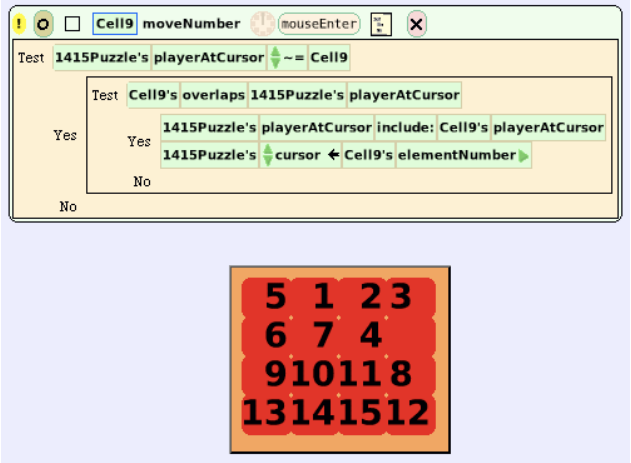

Figure 1. 14-15 Puzzle composed out of four lines of code

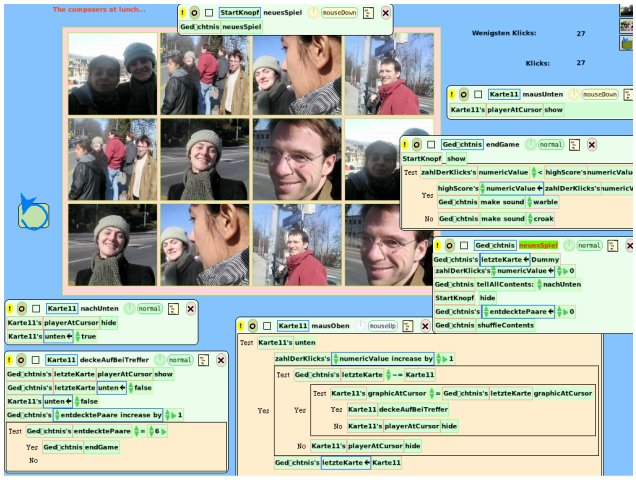

Figure 2. Concentration composed with 32 lines of code

all numbers from 1 to 15 into the right order. As you can see in Figure 1 one can create a running user-interface of this puzzle with only four lines of code.

Features This is a very basic implementation of the puzzle but comes with an easy-to-use interface. The player only has to move the mouse over a stone, which is a neighbor to the empty cell, and immediately this stone is moved to the empty cell.

Idiom: Visual Cursor The actual empty cell gets stored into the cursor of the puzzle-playfield.

Idiom: Connected Neighbors The trick is to let the cells have rounded corners. Then we can change the layout of the cells in such a way that horizontal or vertical neighbor-cells overlap whereas diagonal cells stay apart. The test if a cell is the neighbor of the empty cell then simply boils down to the built-in query whether the cell morph *overlaps* the cell morph of the empty cell stored as a variable in the container.

2.2 Concentration

Concentration [4] is a classic children's game in which the goal is to identify pairs of identical cards amongst a bunch of face-down cards. When a matching pair is found, the cards stay face up until all pairs are found.

Features Our version includes a counter of how many clicks the user needed to detect all pairs, and a restart button which gets activated when all the cards are turned over. As one can see in Figure 2 we used photos rather than symbols, as it is easy to import them into Squeak: Just drop them on the Squeak Window. We composed this game with only 32 lines of code.

Idiom: Visual Cursor The last revealed card gets stored into the cursor of the concentration-playfield. The next time a card is turned it is compared to the most recently flipped one. If they match, both are turned face up, else both stay face down.

2.3 Lights Out

The goal of Lights Out [11] (see Figure 3) is to turn all cells to the same color. If a cell is clicked, that cell and all its horizontal and vertical neighbors invert their color.

Features We only implemented the basic feature which drives the game. There is no score management or reset of the application. The size of the playfield can easily be adjusted by adding or removing cells.

Idiom: Visual Cursor The latest cell pressed is stored in the cursor of the playfield after having been pressed.

Idiom: Connected Neighbors The idea is again to use rounded corners and change the layout so that all horizontal and vertical neighbor cells overlap. Then the second script is executed which inverts all cells overlapping the "currentCell" stored in the container. As the current cell overlaps with itself, it also gets inverted.

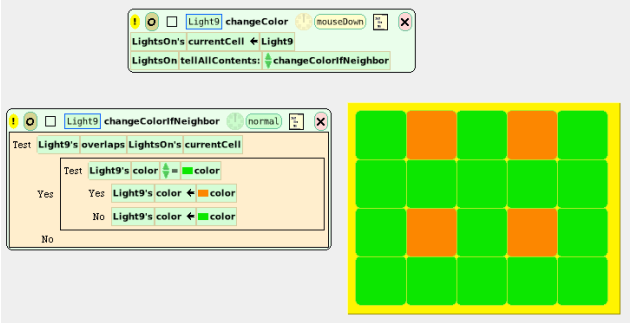

Figure 3. Lights Out in six lines of code

2.4 Pong

Pong (see Figure 4) is the first arcade computer game ever and was originally implemented on an oscilloscope. Two ping pong paddles on either side of the playing table can be moved up and down to bounce the ball back to the other player.

Features We only implemented a very basic ball bouncing algorithm: If the ball hits the paddle, it bounces back with an output angle equally to its input angle. The score of the left player increases if the ball is missed by the right player and vice versa. Nothing special happens in this version when a player wins. The players have to hit the keyboard each time they want to move the ball.

Idiom: Intelligent Environment Both the horizontal and the vertical walls are distinct rectangle-siblings. If a vertical wall finds out that it overlaps the ball, the opposite counter is incremented by one and the ball is put back in the middle of the playfield.

Idiom: Text as variable The counters are actually just texts: Each vertical wall has an instance-variable which is typed as "player" and points to the text.

2.5 Tic Tac Toe

Tic Tac Toe (see Figure 5) is a classical board game and was one of the first programs the first authors developed on the Commodore 64 using Basic, the immediate programming language any owner of the C64 was confronted with after starting the computer. The goal is to to put three stones in a diagonal, vertical or horizontal row.

Features The EToy version provides a fairly complete user interface. It gives the player direct feedback which player won the game and why (indicating the winning line

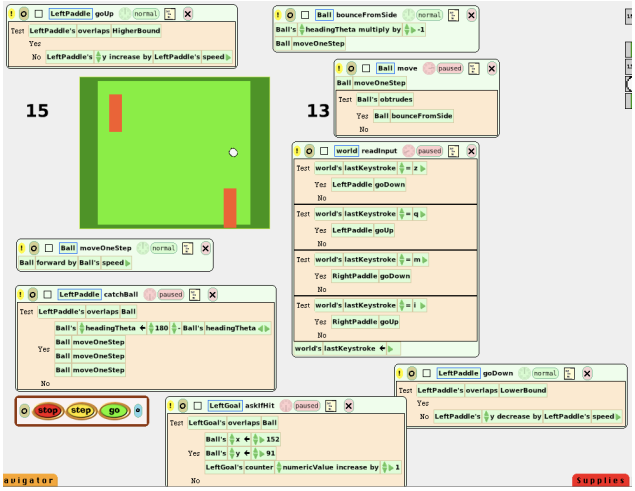

Figure 4. A Pong game composed out of 28 lines of code

in red), and a restart button, which only gets activated at the end of a game.

Idiom: Intelligent Environment Since EToys provides no mechanism to perform matrix queries, we need another way to detect if three stones from the same player occur in a row. We do this by introducing lines as can be seen in Figure 5. We associate to each player's stones either the value 1 or -1. When a stone is placed, its value is added to the counter of each line that it touches. When some line reaches the value 3 or -3, we know that one player or the other has won.

Idiom: Visible Factory We indicate the current player by setting the cursor of the holder to either the circle or the cross. At the same time we use this currently indicated stone as a prototype for copying it into the clicked empty cell.

2.6 Space Invaders

Peter Vogel, a high-school teacher with some background in Basic programming, developed this classic arcade game (see Figure 6) in EToys, with some help of one of the authors.

Features We implemented some bomb-dropping aliens, a ship which can shoot one visible rocket at a time, and which is only steerable via a joystick and not via a keyboard. (keyboard support only came in a later version of EToys). The game comes with a score display and the possibility to restart it.

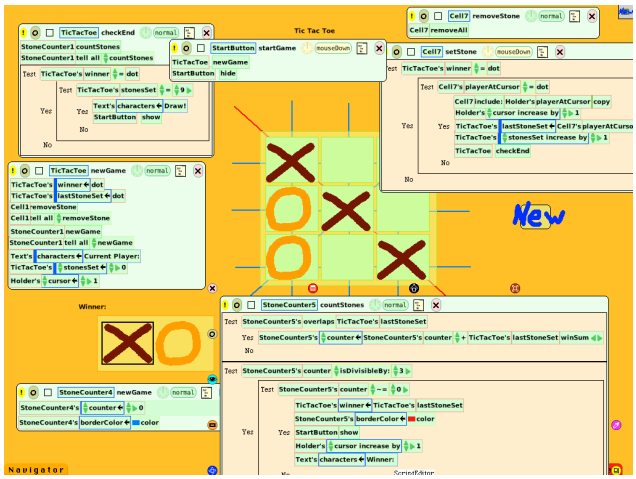

Figure 5. Two-player version of Tic Tac Toe in 36 lines of code

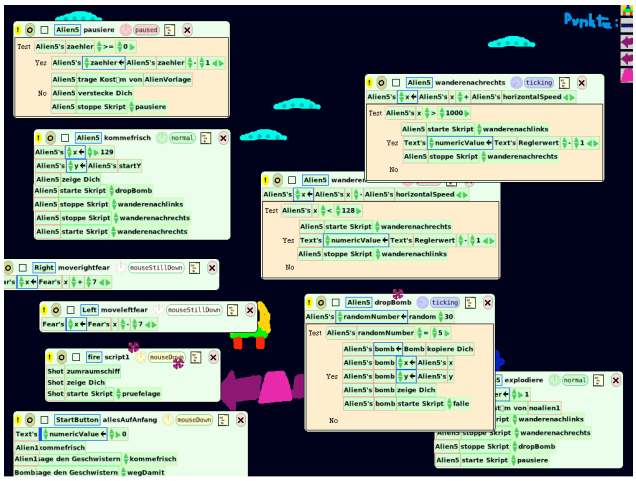

Figure 6. Space Invaders in 64 lines of code

Idiom: Encode interactions into the affected objects When a rocket hits an alien, the alien gets destroyed. To have a reference on the alien we encode the "check for destruction" method on the aliens, which are all siblings. On the other hand we can encode the behavior of alien bombs destroying the mothership into the bombs as we only have one mothership and thus its reference is clear.

Idiom: Text as variable We do not store the actual score in some variable but put it directly in some visible text-field.

2.7 PacMan

Pacman is the classic arcade game in which one has to steer pacman through a labyrinth in order to eat pills and avoid monsters. If certain pills are eaten, pacman can eat monsters for a while.

Features Pills can be eaten, score is maintained, and monsters send pacman to the graveyard. Currently pacman cannot eat monsters. (See Figure 7.) It has to be admitted, that here the fun was more in developing the game than in actually playing it, as the current version is too feature-poor to be really interesting.

Idiom: Intelligent Environment One trick we used here to make the monsters follow the alleys was to insert some points in the crossings. If a monster collides with such a point, it changes direction randomly.

We also introduced a graveyard-playfield: If pacman dies, it gets included into a hidden playfield called "graveyard". Thus we only have to put it back from there when the game is restarted.

Idiom: Encode interactions into the affected Pacman eats pills. But we do not encode this into pacman but into the pill: only then we have a reference to the eaten pill that we can use to hide the pill.

Idiom: Text as variable Again we increment directly a text-field storing the high-score each time a pill gets eaten.

2.8 PetitPetri

Petri nets are a "pinball game" for mathematicians [15]. They are used to formally describe concurrent processes like the classic example of the deadlocking dining philosophers [5]. We implemented a simple Petri net editor in EToys (see Figure 8) and use it in our concurrent programming course for masters students at the University of Bern.

A Petri net consists of transitions and places connected by directed arcs. A place is a container that can hold several tokens. If all incoming arcs to a transition are connected to places holding at least one token, then the transition is *enabled* and can be fired. When a transition is fired all its incoming places reduce their number of tokens by one and all its outgoing places increment their number of tokens by one.

Our implementation [12] makes use of the connectors framework from Ned Konz [10], which comes with the actual squeakland version 05. In contrast to other graphic frameworks like Hotdraw [3] connectors and their connected elements can be programmed directly: Whereas in Hotdraw one usually duplicates the topology of some graph

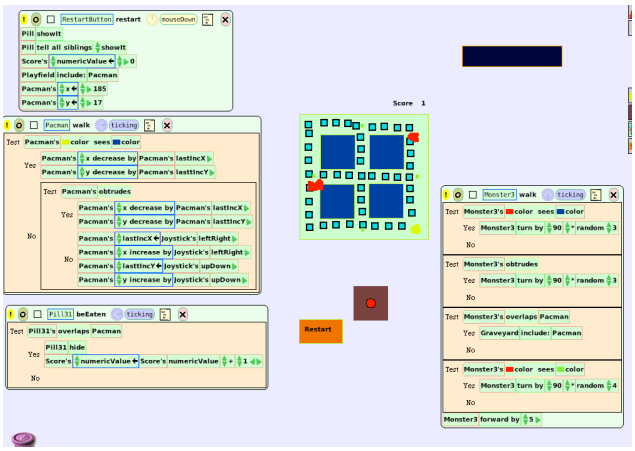

Figure 7. Pacman in 36 lines of code. One cannot yet eat monsters though.

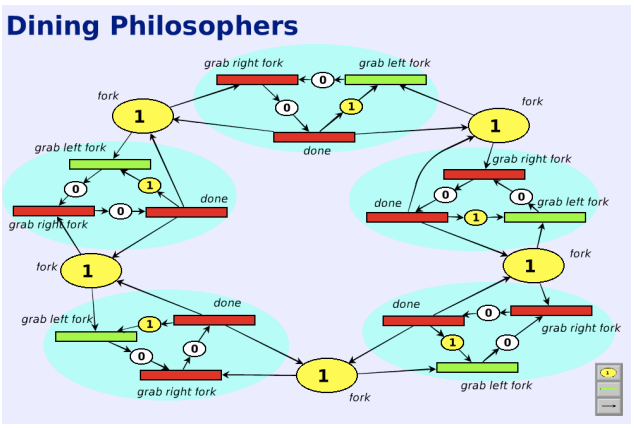

Figure 8. Dining philosophers in a Petri Net implementation consisting of 24 lines of code

by representing the graph both graphically and as a model, connectors invite the developer to directly manipulate the nodes and edges. *Connectors e.g.,* provide expressions to send some messages to all incoming or to all outgoing nodes.

Features This is a very basic implementation of the Petri nets but comes with an easy-to-use interface. Transitions, places and arcs can be created using factory buttons as provided by the connectors framework. If a transition becomes enabled, it turns green, and can be clicked to fire it. Tokens are represented as integers, which can be edited directly. Multiple connections between places and transitions are not provided yet.

Idioms used

Visible Factory An essential part of the user interface of PetitPetri is the button bar, with which new containers, transitions and arcs can be created. The button maker is part of the connectors framework

3. Idioms

Let us now take a closer look at the idioms we used to build the games described above. Although we do not wish to belabor the point whether these are "just idioms" or "really design patterns" [8], we note that they seem to fit the characterization of design patterns posed by the architect Christopher Alexander:

> Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice. [1]

The design principle behind all our idioms is: "Form is function" [19]: The objects introduced or tweaked are all visual and thus merge their behavior and their visualization like the naked objects from Pawson *et al.* [14]. The behavior of the visual objects depend on their form, position or color. Thus we can keep the number of lines of code low. As the architect Frank Lloyd Wright said:

> Form follows function — that has been misunderstood. Form and function should be one, joined in a spiritual union.

3.1 Intelligent Environment

Encode otherwise implicit behavior of an object into another object of the environment.

Motivation Tic Tac Toe is won when three stones of the same kind are put on a horizontal, vertical or diagonal line. Lacking abstract and invisible data types in EToys, the developer is forced to think about a visual representation of these lines. By introducing real and intelligent lines, this problem can be solved. Furthermore the winning condition can be visualized by coloring the winning line red.

Putting intelligence into the environment is a known technique and can be found in Kedama and StarLogo [13] [16] where intelligence is put into the patches in which the turtles live.

Applicability Use this idiom whenever the behavior of an object depends on its environment. Ask yourself if you can alter the environment in such a way that it visually represents the problem.

Consequences Altering the environment by either changing its visual properties, like its color, or by introducing new visual objects not only helps to create a solution in a more distributed and object-oriented way, it often also gives the user of the application direct visual feedback.

Known Uses In the classic EToys car demo, two 10-year old girls came up with a different solution to automatically steer the car around a given track: Instead of providing two different colored sensors they encoded the solution in the environment — when the car sees a blue lake inside the track it turns left, when it sees the gray surrounding it turns right.

In the pacman implementation, we encoded the junctions for the monsters by putting squares on the junctions. When a monster encounters such a square it changes its direction randomly.

3.2 Encode interactions into the affected objects

If an object of one kind interacts with an object of another kind, encode the behavior in the affected object.

Motivation Rockets destroy aliens. To destroy the alien one needs a reference to the affected alien after an encounter with a rocket has occurred. EToys currently do not provide references to the objects touched by a given object. So it becomes necessary to detect the encounter on the side of the affected object, as it is guaranteed to have a reference to itself.

Applicability Use this idiom whenever you need to check for some external condition like collisions and need to change the state of an object later. Encode the test and consequence into the affected object.

Consequences Using this idiom slows down your game. Instead of only letting the pacman check for collisions with pills, one needs to spawn a different process for each pill in the game.

In the case of collisions each EToy should have a pluggable behavior or trait [17] that would allow the developer to ask it for all objects which actually collide with it. Therefore it would be desirable to enumerate these objects and send some messages to them: EToyers miss the `select:` and `do:` idioms known from Smalltalk and this idiom only provides a trick to circumvent this problem.

Known Uses Pacman eating pills and rockets killing aliens.

3.3 Visual Cursor

Store a selected element of a playfield in a cursor

Motivation As we have seen in Lights Out or the 14-15 puzzle, it is often necessary to iterate over all elements of a playfield and compare each element with a given one. To accomplish this we store the element of interest into the cursor of its surrounding playfield.

Applicability Use this idiom whenever you need to select some element of a playfield.

Consequences Actually there is already a cursor in playfields but it is only settable via the index of an element. Incrementing the cursor index of playfields is enough when one wants to do simple animations. For selecting given elements it is rather cumbersome to first detect the index of the element only to set it later. It would be nice if EToys provided a direct way of setting the cursor to some given element.

Known Uses Visual cursors are used for doing animations as shown by Allen-Conn *et al.* in [2]. We broadened their scope for highlighting an element of a playfield to detect overlapping elements like in the 14-15 puzzle or Lights Out or to store them as in Concentration.

3.4 Text as variable

Use texts to directly store values

Motivation It is cumbersome to introduce integer typed variables holding some score-value, only to update some text fields later whenever the score has changed. Instead we use the text-morphs provided by "supplies" to directly store the score into them. We then can get or set the score via the `numericValue` tile found in the "basic"-category of the text-viewer.

Applicability Use this idiom whenever you display some textual information to the EToy-user that might be recomputed later.

Consequences As an alternative one could also introduce new variables and then drag out watchers to display their values to the user. But these watchers come with a small font and currently one cannot change these font-families and font-sizes.

Text-morphs can be seen as global variables. But this does not hurt too much in the case of the games we presented here: All the objects which are of interest to the user are also of interest to the developer and fit on one screen. Texts can also be "scoped" by putting them into some playfields.

Known Uses We used this idiom whenever we had to display some score or high-score to the user.

3.5 Connected Neighbors

Encode neighborhood relationship of objects by letting them overlap.

Motivation Many board games consist of a matrix of connected cells. In some games the cells are neighbors only horizontally or vertically (*e.g.,* 15-puzzle), in some also the diagonals are neighbors too (*e.g.,* Game of Life). In order to detect if two cells are neighbors, one can change the layout of the matrix in such a way that connected neighbors overlap. If the cells are not neighbors on the diagonals, we make the borders rounded, thus reflecting their relationship both to the EToy-Developer and to the user. The developer then only has to ask if the current cell, possibly stored in a *visual cursor*, overlaps a given (*e.g.,* clicked) cell.

Applicability Use this idiom whenever you need to introduce neighbors to each other.

Consequences This idiom allows the developer to detect if two cells are neighbors.

Known Uses See Section 2 for the 15-puzzle and Lights-Out or [6] for the game of life. The trick in the case of these matrix based playfields is to change the layout of the matrix, so that neighboring morphs overlap.

3.6 Visible Factory

Store the prototype of an object in some place which makes sense to the end user

Motivation Having a display indicating the ships left is essential to the space-invaders player. When a ship is destroyed, the user has the imagination, that the next ship comes out of the home base. But what makes sense to the user, also makes sense to the developer: She could reuse the rocket base as a visual prototype-holder to fetch the next rocket, after one has been destroyed.

Applicability Use this idiom whenever you have stones of different colors to set, ships to launch or pacmen to put into some game.

Consequences This idiom allows the developer to display valuable state information like ships left to the user, while fully exploiting the metaphor of a base station by really taking the next ship from there. Therefore the lines of code decrease, and the game is easier to understand.

Known Uses In the Tic Tac Toe example we indicated the current player in the holder by highlighting his/her stone. This very stone is copied into the cell clicked, when the player makes his move, before the next prototype stone is selected again. The button-makers of connectors are visible factories to create visible factories.

4. Concluding remarks

We have shown that it is possible to implement various classical games in EToys with just a few lines of code. As such, EToys is surely attractive to beginning programmers who are eager to see the fruits of their labours as early as possible.

However, we believe that EToys exposes some other important notions that seem to run against the grain of conventional programming. Most significantly, EToys trashes the conventional wisdom that one should separate the model from the view. All objects have a graphical representation. Furthermore there are no objects that are not graphical. This can be both a curse and a blessing, as illustrated by the programming idioms we have identified. For instance, in order to get around the fact that there are no hidden data structures, one must introduce visible entities that make up for this lack. Instead, however, of simulating data structures with graphical objects, *one programs in a different way* — the programmer is forced to think in terms of what is explicit and visibly present.

Some of the idioms we have identified begin to look like design patterns — best practices that that solve common design problems — but others look a bit like clumsy workarounds that compensate for shortcomings in the programming environment. We do not yet have enough insight to propose alternative solutions, but we hope that we may have convinced the reader that EToys are more than just toys, and could well indicate a path to more expressive and direct way of programming.

We also hope our examples and suggested idioms help a bit to make teaching and learning to program possible and fun, which it should be.

Acknowledgments

We gratefully acknowledge the financial support of the Swiss National Science Foundation for the projects "A Unified Approach to Composition and Extensibility" (SNF Project No. 200020-105091/1, Oct. 2004 - Sept. 2006), and "RECAST: Evolution of Object-Oriented Applications" (SNF Project No. 620-066077, Sept. 2002 - Aug. 2006).

References

[1] Christopher Alexander, Sara Ishakawa, and Murray Silverstein. *A Pattern Language*. Oxford University Press, New York, 1977.

[2] B.J. Allen-Conn and Kimberly Rose. *Powerful Ideas in the Classroom*. Viewpoints Research Institute, Inc., 2003.

[3] John Brant. Hotdraw. Master's thesis, University of Illinois at Urbana-Chanpaign, 1995.

[4] Concentration. http://en.wikipedia.org/wiki/-Concentration_(game).

[5] Edsger W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1(2):115–138, 1971.

[6] Composing Simple Games with EToys. http://www.emergent.de/etoys.html.

[7] 15 Puzzle. http://en.wikipedia.org/wiki/N-puzzle.

[8] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, Mass., 1995.

[9] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: The story of Squeak, A practical Smalltalk written in itself. In *Proceedings OOPSLA '97, ACM SIGPLAN Notices*, pages 318–326. ACM Press, November 1997.

[10] Ned Konz. Connectors: A framework for building graphical applications in squeak. In *IEEE C5: The Second International Conference on Creating, Connecting and Collaborating through Computing*, volume 2, pages 96–103, 2004.

[11] Lights Out. http://en.wikipedia.org/wiki/Lights_Out-_(game).

[12] Oscar Nierstrasz and Markus Gaelli. PetitPetri—A Petri Net Editor done with EToys, September 2005. http://www.iam.unibe.ch/∼scg/Teaching/CP/PetriNets.

[13] Yoshiki Ohshima. The Early Examples of Kedama, A Massively Parallel System in squeak. In *IEEE C5: The Third International Conference on Creating, Connecting and Collaborating through Computing*, volume 3, pages 93–100, 2005.

[14] Richard Pawson and Robert Matthews. *Naked Objects*. Wiley and Sons, 2002.

[15] James L. Peterson. Petri nets. *ACM Computing Surveys*, 9(3):223–252, September 1977.

[16] Mitchel Resnick. *Turtles, Termites, and Traffic Jams*. MIT Press, 1994.

[17] Nathanael Schärli. *Traits — Composing Classes from Behavioral Building Blocks*. PhD thesis, University of Berne, February 2005.

[18] David Ungar and Randall B. Smith. Self: The power of simplicity. In *Proceedings OOPSLA '87, ACM SIGPLAN Notices*, volume 22, pages 227–242, December 1987.

[19] Bosse Westerlund. Form is Function. In *Proceedings DIS 2002 Serious reflection on designing interactive systems*, pages 117–124, July 2002.

[20] David Wheeler. http://en.wikipedia.org/wiki/-David_Wheeler.