

Modular Morphic Super Chalk Board

Embracing Wabi-Sabi

Gary Dunn
Open Slate Project
<http://openslate.org>
osp@aloha.com

May 2011

Epiphany

Super Chalk Board was to be the defining application for the Open Slate Project. The software by which the bulk of classroom activity would take place, whether in a real room (a.k.a. brick and mortar) or over the Internet. Originally I thought of Super Chalk Board as a single application. A PowerPoint on steroids, HyperCard meets Jabber. A Swiss Army Knife of virtual learning.

Recently I was playing with the new Squeak 4.2 image and had an epiphany. *Just use the resources already there.* The more I look, the more it seems that most of the functionality I wanted already exists in Squeak, only in a Lego sort of way. In place of a single, monolithic application we can use building blocks which can be combined to perform the same tasks. Blocks that can be combined in many ways, to suit the needs of the author, the teacher, and, most important of all, the student. The result: a much more flexible solution, one which extends the learning environment into the process itself.

The most significant difference between my old and new vision of what Super Chalk Board should look like is that this new, modular version treats all participants as creators. This is consistent with other OSP principles – students should build their own computers, run the school network, develop course content and mentor younger students. Working with a modular Super Chalk Board is not as simple as getting a soft drink from a vending machine, but it is no harder than baking toll house cookies.

In Japanese culture there is a concept called *wabi-sabi*. It is a value of which things can have more or less of, like beauty, but it has more to do with function than appearance. Wabi-Sabi is said to express the notion that “nothing lasts, nothing is finished, and nothing is perfect.” This expresses perfectly the current state of Squeak and in particular its collection of building blocks called morphs. They are by no means perfect, yet even so have much to offer, including the lesson that nothing is perfect.

Projects

Squeak organizes work into projects. Visually, a project resembles a tabletop onto which can be placed any number of programmatic building blocks. A Squeak image can have many projects, any one of which is active. The active project fills the Squeak world, the virtual desktop within the Squeak application window on the host operating system. The user can flip through the available projects or jump to whichever project they choose.

Morphs

The programmatic building blocks used in a project are a mixture of new and pre-composed blocks. Most of what Chalk Dust needs can be accomplished through clever use of existing blocks. Squeak calls these programmatic building blocks morphs (from the Greek word for form or thing). A project built from them is called a morphic project, as distinguished from an mvc project, which is built on the older, more conventional Smalltalk behavior.

The following table lists some morphs included in the standard Squeak image that are likely candidates for Chalk Dust projects:

Text	Rectangle	Storyboard
Text (border)	Polygon	Stack
Scrolling Text	Ellipse	Book

The morphs in the table above, along with many more, are available on a slide-out flap labeled “Objects.” To add one to the current project, click on the Objects tab, navigate through the flap to locate the desired morph, then drag it out and drop it onto the project desktop. Give it a name, move it around, re-size it, change its color. It is simple, intuitive, and familiar to anyone who has worked with vector graphics applications.

Saving and Loading Projects

A project can be saved to disk separate from the Squeak image. Squeak wants to save projects in a folder named Squeaklets, located in the folder containing the current image. The Save Project dialog also allows projects to be stored elsewhere in the file system, but having a consistent, standard location works to our advantage.

A saved project is written out in a single file. The contents of the file are references to the morphs used in the project, and all the data associated with those morphs. What this means is that we can expect our projects to migrate easily into newer versions of Squeak, because the code that they rely on comes from Squeak rather than the project file. Note that new morphs, or substantive changes to standard morphs, require those object classes be saved separately; such changes are not included in a project file. The Monticello code and version control tool can be used for this.

When a saved project is loaded the effect is the same as creating a new project. The loaded project is not merged into the current project, it gets a new world. The project is merged into and saved with the current image.

The save command implements a basic form of version control. When saved, a project named “notepad” will be stored in a file name “notepad.001.pr.” When saved again it will automatically be stored in “notepad.002.pr.” The load command lists the full names, so it is possible to revert to an older version. The important thing is that it is hard to accidentally overwrite a project.

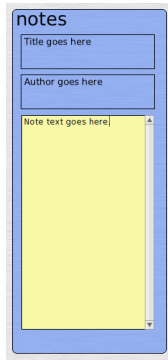
Distributing Projects

A project file is just a data file. It can be distributed just like any other data file, using whatever methods are appropriate to the situation.

Imagine a project designed as a lesson combined with a student worksheet. The student receives a fresh copy of the project, reads, watches or listens to its content, then interacts with the practice work. Instead of turning in a worksheet the students turn in a copy of their project file. If the student copy gets hopelessly mangled it can be renamed or thrown away and a new one loaded.

A Simple Notepad

My first example combines several basic morphs to create a place for taking notes. The result looks like this:



The steps to make it are as follows:

1. Click on the desktop to bring up a World menu, then select “projects”
2. Click on “create new morphic project”
3. Click the name at the bottom of the new project’s thumbnail and change the name to “notepad”
4. Click inside the thumbnail to enter the new, empty notepad project.
5. If the Objects flap is visible at the top of the window, skip to step 10

6. Pull down the Apps menu – or – click on the desktop to display the World menu and select “open”
7. Select “Preferences Browser”
8. Select the category “flaps”
9. Enable “showSharedFlaps”
10. Close the Preferences Browser
11. Click on the Objects flap
12. Click on the “Graphics” category
13. Drag a RoundedRectangle onto the project desktop
14. Blue-click on the RoundedRectangle to bring up its halo
15. Click at the bottom, on the text “RoundRect,” type “Notes” and press Enter
16. Blue-click on the morph and drag the yellow button to re-size the rectangle
17. Blue-click on the morph, click the purple (eye dropper) button and select the desired color
18. The next three morphs come from the Text category. To avoid having them disappear behind your notepad, drop them along side, customize, then assemble on top of the notes morph.
19. Drag a “Text” morph onto the desktop
20. Rename it “NotesLabel”
21. Click on the content “Text” and change to “notes”
22. Select the text, then blue-click the morph and click on the yellow button to re-size the text morph so that all of the letters appear on the same line
23. Select the text, then click on the light-brown halo button (FF) and select “Normal” to clear the default bold setting
24. Blue-click the NotesText morph and carefully use the black halo button to move it into position on top of the Notes morph.
25. Click on the red halo button (menu) and select “embed into,” then from the sub-menu select “Notes”
26. Bring up the red halo menu and check “resist being deleted”
27. Bring up the red halo menu and check “resist being picked up”

28. Bring up the red halo menu and check “be locked”
29. Repeat the process to add the title box, using a Text (border) morph; use the light-green halo button (FF) to change font family as size - use BitstreamVeraSans 12pt.
30. Now add a Scrolling Text morph, name it “NotesScrollingText” and set the font to Bitmap DejaVu 9 pt.

You now have an attractive place to take notes. Whatever is written here will be saved when the image is saved, so there is no need for a save command. Before you start using your new notepad, save it as a project. That way, whenever you want a fresh, empty notepad just load the project.

When you do want a fresh notepad you probably will not want it all alone on the desktop. You can use the “Squeak” flap along the left edge of the desktop to move morphs between projects. Load in a new notepad project. Click the Squeak flap, drag the notepad onto the flap, go to the project where you want it, and drag it out onto the desktop.

Take It Up a Notch

Our notepad can do more than store text. Notes can be illustrated with a hand drawn sketch. Here’s how.

Go to the Graphics category on the Objects flap and drag out a Paint morph. This is a simple paint program designed for younger artists. It lacks the sophistication of The Gimp, but it does what we need. Draw a little picture, and finish by clicking on the “Keep” button.

You now have a paint object floating over your world’s desktop. The next step is to attach your picture to the notepad. Do this the same way you did the text elements, by embedding the picture into NotesScrollingText. (By now it should be clear why you want to name your morphs.) At this point the picture is sitting on top of the text, but it will not move with the text when it is scrolled up and down. To enable that is easy. Begin by moving the insertion point to where you want the illustration to appear. Blue-click the picture, click on the red button, select “text anchor” and from the sub-menu select Inline. (“Text anchor” will not appear on the menu unless the object is embedded into a text morph.)

You can also add a star, or an ellipse, or a lot of other things, using the same procedure.

Live Session

One of the more ambitious goals of Super Chalk Board is the ability to stream live updates to all participants. In a typical instance, the instructor will prepare a page and mark it up while speaking. Or, even easier, the instructor will begin with a blank “chalk board” and write on it as the session evolves. Adobe Connect is a good example of what I originally envisioned. Super Chalk Board

was supposed to stream these live events to the participants slates, so that everyone sees and captures what the instructor writes, automatically. For a time I thought the vector drawing program Inkscape could be used for this purpose, but its networked mode never panned out.

Squeak has a package called Nebraska, "... a toolkit for building remote interactions with Morphic in Squeak." (<http://wiki.squeak.org/squeak/1356>). There are many ways Nebraska can be used to make a Chalk Dust application collaborative. To implement the original live stream concept the instructor designates their computer to be a server. The students in the class connect to the server, and everyone is able to see and manipulate the shared world.

One limitation of the Nebraska package is that it does not work well beyond a LAN. It cannot be used as-is with a typical Internet connection, involving a NAT-based router, because connections are based on IP addresses. Modern firewall configurations also interfere with Nebraska due to its use of ports.

To better understand how sharing in Nebraska works, and its limitations, let's take a look at a typical session. Before becoming a server the instructor must know the IP address of their computer, so that they can share this with the clients. The instructor should set up a desktop with only the morphs to be shared. To become a server, first drag a Nebraska Server button (yellow, labeled "Share") from the Objects menu, Collaborative category, onto the desktop. Click on the yellow button and a tile labeled "Nebraska 0 clients" will appear at the upper left corner of the screen. To stop sharing, click the "X" at the left edge of the tile.

To connect to this share requires a badge representing this computer. Such a badge is on the Objects menu, but that one is intended to represent the local computer. What a client needs is a badge for the server. The most straightforward way to create such a badge involves dipping our toes into a little Squeak programming. This needs to be done on every client.

```
EToySenderMorph
```

```
new
  userName: 'Gary'
  userPicture: nil
  userEmail: 'knowtree@aloha.com'
  userIPAddress: '10.0.1.4';
  position: 200@200;
open
```

1. Open a Workspace. There are several ways to do this, try clicking on the desktop to bring up a World menu, then select "Workspace."
2. Enter the code shown above; substitute the name, mail address, and most important, the IP address of the server.
3. Select the entire statement.
4. Yellow-click the text and from the pop-up menu select "do-it." A pink badge will appear filled in with the information supplied.

This process of writing some code in a workspace and executing it is commonplace to Squeakers. In this case, the code fragment is analogous to what we might enter on the command line in Unix, a command to run and some parameters for the command. This example sends the message “new” to the class EToySenderMorph, the pink badge thing. This creates a new instance of a badge, although we do not see anything yet. The next message is a long one in four parts, sent to the new instance. The position statement tells the badge morph where to place its upper left corner. Without this statement the badge will appear at 0@0, the upper-left corner, which is not a mistake. The last message, “open,” displays the badge morph.

The badge we just created has a row of buttons along the bottom, labeled C T ! ? A S.

C - open a text chat session with this person

T - start telemorphic with this person

! - tell this person about the current project, where the project file is located

? - see if this person is available

A - open an audio chat session with this person

S - see this person’s world

Click on the “S” and, provided they have activated their yellow Share badge, the server’s desktop – its world – will appear in a window. Every mouse pointer of every client will appear in the window, labeled with its value of userName.

Collaboration

Any slate running Squeak can become a server. This opens up many opportunities beyond the traditional lecture. The teacher can divide the class into groups, with one member of each group becoming the server for their group. At the end of the breakout session each group can share their results with the class by sharing their group server with the entire class, or by having each group upload their results to the teacher’s computer and present from there.

A not-so-obvious feature of the badge is the ability to copy morphs to the badge owner’s computer. In our example, each student has a badge representing the teacher’s computer, and by clicking the "S" button they share the teacher’s world. A badge can display an avatar to visually identify who it represents, or if undefined, a large question mark. Any morph dropped onto this part of the badge will be copied to the badge owner’s world. The other computer does not need to be in server mode for this to work. Our notepad is an example of just such a morph. Technically it is a group of morphs, yet even so it can be sent through the tunnel and a perfect copy come out at the other end.

Mentoring

A slightly different kind of live session activity is provided by a Squeak feature called telemorphic. A good example of telemorphic in action is a teacher tweaking a student project. When the teacher initiates a telemorphic session with a student, the teacher’s mouse pointer appears on the student’s world and the teacher sees the student’s world and can interact with it while the student watches. The greatest difference between sharing a world and telemorphic is

that sharing is a group activity in which everyone sees or manipulates a common world, while telemorphic is one-on-one, where one person manipulates another person's world.

Communication

Three Nebraska morphs support communication. Two are for text messaging, and one is for audio chat. They work, and may be useful in ways not foreseen here, but due to the difficulty in using Nebraska outside the classroom their utility is limited. I suppose text chat could be a way to reduce classroom noise!

The Presence Issue

The biggest obstacle to using Nebraska is hooking up – discovering who is on the network and what their IP address is. For sharing this is less of an issue because only the server IP address need be known. Any client in the room can connect as long as they have the server's address.

Person-to-Person interactions require that all participants know the IP addresses of everyone else. Since in a typical DHCP setting these will change from day to day, setting up even a half dozen badges will take some time.

The Sugar environment used on the One Laptop Per Child project solves the presence issue using XMPP. Essentially, all participants connect to a specific Jabber server. We could use a similar solution, but that would require modifications to Nebraska or even new code, which at this point we want to avoid.

Working With Nebraska's Limited Scope

When OSP was created the Internet was not such a bad place. Students working off-campus would connect to their ISP via a dial-up connection. Firewalls blocked a few evil IP addresses and allowed everything else through. Nebraska worked well in this environment. These days, the combination of constant attacks, high-speed connections and NAT routers at home make using Nebraska outside the classroom a security risk as well as technically unfeasible. In a friendly, co-located environment, such as a classroom, Nebraska has much to offer.

Web Browser

A practical way to extend the reach of Squeak projects would be to place them on a web server. Squeak has a web browser (Scamper) but it lags far behind the level of development required to function as a modern web browser. On the other hand, any host operating system will have several excellent web browsers to choose from, and the user is likely to be proficient with at least one of them. Once downloaded, a project file can be loaded as described above in the section "Distributing Projects."

Jabber

Any good Jabber (XMPP) client can do file transfers. Students could use existing Jabber servers to hook up when not in class, collaborate using text chat and share projects under development. Not as cool as sharing a world in Nebraska, but still useful. A more fully developed solution would be to set up a Jabber server for a school, or a group of schools.

Next Steps

Continued progress will require effort on three fronts:

Technical As I write this the 4.2 release still has serious bugs in Nebraska. Sharing in particular is broken. I believe the fix is easy because it was all working perfectly late in the 3.x series. Currently a great deal of effort is going into the introduction of the cog VM. When this is complete we can expect – or at least hope – some resources to go towards fixing Nebraska.

Examples We need to develop a set of example projects that can be copied and used to develop actual projects. In particular we need to develop instructions on how to make effective books and stacks. I am currently working on a morphic book that explains how to make a morphic book.

Content No matter how appealing the Squeak environment is, it will not fulfill the goals of Chalk Dust without content. To this end we need to attract authors. Hopefully, having a well designed, fully functional platform will help us in that endeavor.

This work will be much easier with more people contributing to OSP. I wish we had the problem of organizing a large group of volunteers. Recruiting new members remains a top priority.