Computers & the Humanities 281

Labeled Drawing Task

In almost any given beginning-level language book, you find several pictures that have been labeled in the target language: a pictue of a room, a drawing of a person, an image of a specific object, etc. Such pictures are designed to introduce vocabulary words in context. Through judicious application of the various techniques we have acquired in using Revolution, we can create a similar activity to introduce vocabulary to a new language learner. With computer technology, however, we can make the drawing a little more interactive.

Beginnings

First, we need to start with a drawing, which must be scanned in and placed on a card. We want to give target language terms for items in the scene. We can achieve that by creating a label field for each item which would appear when the user clicks on that part of the drawing. For example, if we're working with a classroom and we want to label the window, we could create a field and set its name to "window" and place the word "window" in the field (in the target langauge, of course). Then we would create a transparent button, name it "window," and position it over the window in the image. Arbitrarily giving the field and the button the same name gives the two objects an identical property value that we can exploit for this particular activity.

We now need to create a mouseUp handler for the button's script, something similar to this:

```
on mouseUp
   show field "Window"
end mouseUp
```

This will essentially make the window in the image "hot," enabling users to click on the window and learn the term for window in the target language. We can also add a handler to the script of the field so that it the user may make it disappear by clicking on it:

```
on mouseUp
  hide field "Window"
end mouseUp
```

Both handlers give the user control over both showing and hiding the label for the window in the drawing.

Now the same process needs to be replicated for each item in the picture that we wish to label and make clickable. This is easily accomplished by duplicating the various button and field objects and changing the names, sizes, contents, and handlers appropriately. One way to avoid having to change the script for each field could be to make a more general handler to hide the object that would work for each object, something similar to this:

```
on mouseUp
  hide me
end mouseUp
```

This handler accomplishes the same purpose as the previous handler, but has the added benefit of applying to any of the label fields. If this were the handler in the script of the original field, each duplicated field would already have a working script that would not need to be altered.

A Question of Efficiency

Since we've tackled the problem of duplicating effort with the fields, let's see what we can do with the buttons. A close examination of the buttons and their handlers reveals that they are all essentially the same in their function: Each button hides a field that has a name that corresponds exactly to that particular button. The only variable in this process is the name of the buttons. This is usually a sign that we need a more general handler on the card script to handle this task. We just need to somehow get the name of the button clicked, then show the field with the same name. Remember that in Revolution `the target` function identifies the object which the user clicked and which first received the mouseUp message. We can grab the short name of the target and put it in a variable and then use that variable to identify which field needs to be shown. Our handler would then look something like this:

```
on mouseUp
   put the short name of target into which
   show field which
end mouseUp
```

This effectively takes the name of the button and then shows the field that has that same name (facilitated by the fact that we have designed our activity to have corresponding buttons and fields with the same name). We have to use `the short name`, or we would get its full name: `button "itemName"`. If we don't put any handlers in the scripts of the buttons, then the mouseUp message goes to the card with this handler script which takes care of all clicks in the buttons.

With this design we can create as many buttons as we need. We just need to ensure that they are named properly and layered appropriately (buttons for small foreground items would be in front of buttons for larger background items). To cover items with irregular shapes in the drawing, we can use multiple buttons to approximate its shape (it doesn't matter how many buttons we use for any given item in the picture as long as they all have the same name). Now all our objects in the illustration should be labeled and clickable.

With the design as it presently stands, we don't have to worry about the fields because they each have a mouseUp handler which intercepts the message and prevents it from moving on to the card script. However, clicking on the card will create an error message as Revolution is unable to find a field with the same name as the card. This would need to be fixed.

Also, we encounter here again the issue of general scripts. Since the script for each field does essentially the same thing, we have another candidate for a card level handler. However, we already have a mouseUp handler in the card script. How could we then tell the difference from a mouseUp in a button and a mouseUp in a field? Remember that the name of the target will be `field "thisfield"` for fields and `button "thisButton"` for buttons. Knowing this we can check in this fashion:

```
on mouseUp
   if the name of target contains "button" then
      put the short name of target into which
      show field which
   else
      hide target
   end if
end mouseUp
```

This handler will check the name of the object clicked by the user. If that name contains the word "button", we then know that it was a button clicked by the user. We can then show the appropriate field. Since there are no other objects than the fields that concern us, then we can assume that if it's not a button it's a field, and consequently hide that field. Thus, we no longer need handlers in individual fields. In fact, with this handler in the card script, we have no need for handlers in the individual card objects. This results in having only one handler to edit/change (if needed) that controls the actions of almost an unlimited number of other objects, a picture of efficiency.

This script also fixes the problem with clicking on the card. However, it creates another problem in that the picture is hidden upon clicking it, a result which undermines the entire activity. One solution to this difficulty would be to insert a second if-statement rather than an else-structure to handle the cases where the target is a field:

```
on mouseUp
   if the name of target contains "button" then
      put the short name of target into which
      show card field which
   end if
   if the name of target contains "field" then
      hide target
   end if
end mouseUp
```

With the handler structured this way, when target is anything other than a field or a button, it doesn't fit either if-statement and is subsequently ignored. The moral of this story is that whenever you implement a mouseUp handler at the card level (or higher), make sure you account for clicks on the card as well as on all other objects.

Shall I Repeat This?

Despite the wisdom of the adage that "if it ain't broke, don't fix it," there is still room for improvement with this labeled drawing. As presently designed, the stack requires the user to click each field to make it disappear. This can be quite annoying, particularly if there are numerous items we've labeled. The picture rapidly becomes cluttered with labels. We need a quick way to hide all fields without having to click on each one. Let's create a reset button. We could script it to hide each field by name, but this would require us to write a hide statement for each labeled item. Obviously this is too much work (remember that it is laziness, er, efficiency that pointed us in this direction in the first place). We need a way to cycle through the fields and hide each one. A repeat loop would suffice:

```
on mouseUp
   repeat with nn = 1 to the number of fields
      hide field nn
   end repeat
end mouseUp
```

This loop takes a variable and initializes it to 1 and then hides the first field. This particular repeat loop increments the variable by one, and so the second field is then hidden, then the third field, and so on. By using `the number of fields` within the control structure for the repeat loop, this will hide exactly the number of existing fields: No more, no less. With the handler written this way, we could add/delete fields and not have the need to alter the handler. It will always cycle through the number of existing fields and hide each one from the first until the last. This gives the user a greater level of control over the learning environment.

However, in solving one "problem" we immediately create three or four new problems. Scripting it this way will hide ALL fields, including information fields we have on the card to label the card, give credit, etc. Since we don't want that to happen, we need to somehow find a way for Revolution to differentiate between fields that need to stay and fields that need to be hidden. One solution is to group the fields you want to remain showing, then to refer to the other fields as "card" fields. This will effectively ignore any fields contained within groups.

Getting the Message

Once again, we don't need to be completely satisfied with our present design, as there is always a better way to do it. Currently the exercise is based entirely on the mouseUp message: The user clicks to see a word and then clicks to hide that word. We could take advantage of the other messages being sent and incorporate them in such a way as to improve the design of our stack. First, let's think about using the mouseDown message in conjunction with the mouseUp. These are reciprocal messages that complement each other nicely for this type of activity. Let's place two message handlers in the card script:

```
on mouseDown
   if the name of target contains "button" then
      put the short name of target into which
      show card field which
   end if
end mouseDown

on mouseUp
   if the name of target contains "button" then
      put the short name of target into which
      hide card field which
   end if
   if the name of target contains "field" then
```

```
      hide target
   end if
end mouseUp
```

This design creates an environment where holding the mouse down over a particular item will display its label. Letting the mouse go up will then hide the label. This effectively displays the label as long as the mouse is down. The second if-statement in the mouseUp handler is to account for a user moving the mouse before letting it go up (remember to account for all possibilities). This is a slick and clean design which renders the reset button obsolete (i.e., the activity cleanup is not user-dependent).

Another possibility is to use the mouseEnter and mouseLeave messages. Again,these are two messages that are complementary and could be used effectively together. Let's try these two handlers in the card script:

```
on mouseEnter
   if the name of target contains "button" then
      put the short name of target into which
      show card field which
   end if
end mouseEnter

on mouseLeave
   if the name of target contains "button" then
      put the short name of target into which
      hide card field which
   end if
end mouseLeave
```

When paired in this fashion, these two message handlers cause the corresponding words to appear when the user passes the cursor over each item in the drawing. Upon exiting the item, the corresponding field is then hidden. Again, this design has no need for a reset button. The user leaves the activity in the same condition in which it was found, so it is alway ready for the next user.

It's a Wrap

Your assignment, then, is to create a labeled drawing exercise/activity of your own.

1. Find a beginning-level language textbook to use as a source. The textbook may be for either college or high school level.
2. Locate a simple picture in that book and scan it (or draw one of your own).
3. Create a stack and entitle it ***YourName*--LabelDraw**. Give the stack an appropriate label (based on the language with which you are working). Place the picture on the first card of the stack.
4. Identify objects in the picture by creating fields to label each part (at least 10 labels).
5. Make each of those objects "hot" or clickable through some means so that the corresponding field/lable is shown appropriately.
6. Put a copy of your finished stack in the **Assignments** folder.

These notes present one method to create a labeled drawing activity for language learning. Hopefully one principle you learned from this exercise is that there is always room for improvement. Also, since what has been presented here does not pretend to be the most effective or efficient method, you are free to improve upon it. This was designed mainly to get you started with the creative process. Use your creativity to expand upon the information given and build something extraordinary.

Course Schedule
Main Page