PsyScript tutorial 3: Growl

This tutorial covers the following topics:

- sound stimuli
- showing and hiding an image
- tables
- random order
- doing many trials with one subject

http://www.psych.lancs.ac.uk/software

Prerequisites

This tutorial won't make much sense unless you've completed tutorials 1 and 2.

Preparation

Find the folder called 'growl tutorial' supplied with PsyScript, and make your own copy of it somewhere where you can make changes.

The experiment

The idea behind this experiment is to present the subject with different pictures and different growls and to find how scary the subjects think they are. You can read the instructions I wrote (in the file `instructions.txt`) and examine the stimulus files to get an idea how it should work.

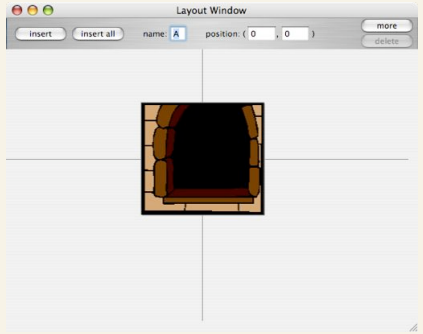
As you did in the other tutorials you're going to start off with a very simple script and gradually add things to it until it conducts a proper experiment. Along the way you'll learn to use several features of PsyScript that previous tutorials haven't mentioned.

Testing the stimulus commands

Eventually you'll have a script which uses all the pictures and sounds in the 'growl tutorial' folder but a good way to start would be to make sure all the stimulus commands (both image and sound) work. The fastest way to set up some image instructions is to use the 'Layout' button.

Start up PsyScript. Since you didn't run it by clicking on an existing script, it will show you a script with no functional commands in. Save this script in your copy of the 'growl tutorial' folder. Then hit the 'Layout' button to bring up the layout window.

Most of the time the subject is just going to see a dark cave – the 'dark.png' image – in the middle of the display. So delete two cells, leaving one cell in the middle of the window (it doesn't have to be the exact middle) and drag the 'dark.png' file onto it:



If you hit the 'insert all' button and close the Layout window you should see the following new commands in your script:

```
move cell A to (0,0)
load image dark.png into cell A

show cell A
```

That's a good start. You can run this script if you like, but without a 'wait' command at the end you won't get very long to see the results.

So add one line just before the `end proc` line:

```
wait for 5 seconds
```

After the subject has been looking at the dark cave for a while you want something to appear in it. Since you already have the A cell showing in the right place it's simplest just to load a different image into it. Eventually this will be randomly chosen from the available animal images but for this bit of testing we'll just pick one. Add this command after the `wait` one:

```
load image beast normal.png into cell A
```

We want the image to be accompanied by a growling sound. Again, we have a number of different ones to choose from but this is still just testing so you can pick any one. Add this command after the one you just added. You can either just type it or use the 'Commands' window, find the new `play sound` command you haven't used before, click the round button next to it, then replace the question mark with the name of the file you want:

```
play sound Lion.wav
```

once the sound has finished playing you want the animal to disappear again. Add this:

```
load image dark.png into cell A
```

then you want another pause to appreciate the dark cave:

```
wait for 5 seconds
```

If you've added all those instructions, your script should look something like this (hit the 'Check syntax' button to get all the indentation right):

```
-- PsyScript script format 1.0

proc main
  move cell A to (0,0)
  load image dark.png into cell A
  show cell A
  wait for 5 seconds

  load image beast normal.png into cell A
  play sound Lion.wav
  load image dark.png into cell A
  wait for 5 seconds
end proc
```

(I added a blank line to split the commands up nicely.) You should now be able to run this (you might want to save your changes first) and see the results. That's pretty-much it: you've tested all the stimulus commands you need for the experiment. Everything else is minor details.

Testing the reponse commands

After stimuli come responses. A good thing to do now is to test the response commands you're going to use.

In this experiment the response should come after the stimulus (the picture of the animal and the sound of it) has both gone away. Therefore you should replace the final `wait` command with something that waits for the response instead of waiting for a fixed amount of time. In this experiment the subject should be hitting one of three keys to rate the scariness. Replace the last `wait` command with

```
wait for a key in qaz
```

and try it out. You should find that all other keys are ignored. Of course, this response isn't logged yet which would make it pretty useless, so add these after the command that waits for a response:

```
log $lastKey
log $lastKeyTime
log $return
```

Now's a good time to save your script in case you lose everything. Test the script.

Top and tail

You've now tested both the stimulus and response commands and it's time to fill out the rest of the commands of the script. It would be a good idea to start off by showing some instructions and by logging the subject's details so add these lines before the first command (after the line which reads `proc main`)

```
display text from file instructions.txt
log $dateStamp
log $timeStamp
log $return
log $experiment
log $subject
log $comments
log $return
```

At the end of the script add this line

```
display text from file thankyou.txt
```

Add blank lines and/or comments to taste. Save the changes and run the script again if you want to see what the experiment is like with instructions.

Showing and hiding

Well, that worked fine except that for some reason I kept staring at the dark cave forgetting I was meant to react to the stimuli. I think we need some instructions on the screen when the computer is waiting for a key.

These instructions would just be another image, stored in just another image file. But I'll want to use it a little differently to previous images: I want to set it up once at the beginning of the experiment, then just show and hide it as required.

First, set it up: move the cell to the right place and load the right image into it. This can happen immediately after setting up the picture of the dark cave: just after

```
move cell A to (0,0)
load image dark.png into cell A
```

add

```
move cell B to (216,0)
load image rateme.pict into cell B
```

however, although the next line says `show cell A` we're not ready to `show cell B` yet: we want B to show only when PsyScript is waiting for the subject to press a key.

That's done in this line:

```
wait for a key in qaz
```

so put one line above it and another below it:

```
show cell B
wait for a key in qaz
hide cell B
```

Now the instructions in cell B will be shown just before the script waits for a key, then hidden immediately after a response is accepted. Save your script and try it out.

I found that the position of the instructions `(216,0)` was a little too close to the picture of the cave to suit me, so I moved it to the right by increasing the `216` to a bigger number.

Previously you always loaded up an image just before you wanted to show it. The image in cell B, however, is loaded at the beginning of the script and cell is made visible and hidden as appropriate. While an image is loaded into a cell you can use the 'show cell' and 'hide cell' commands. You can even use the 'move cell' command to move the cell around.

Picking one stimulus from a list

So far so good: one image and one sound. But most experiments present a number of different stimuli, usually in random order. The rest of this tutorial is about tables. Tables are the way you make up a list of things and let PsyScript either pick items from the list or go through the entire list, possibly in a random order.

This experiment has two things you could use a table for: the sounds and the images. You're going to start off by making a simple table to let PsyScript select one of the sounds to play. This table goes after the main procedure in your script. (A script can have any number of procedures and any number of tables in and they can be in any order. So far all of your scripts have had just the essential 'main' procedure and no tables.)

After the last line of your script (`end proc`) add a blank line, then the following

```
table sounds
end table
```

then move the cursor back so it's just before the first `e` in `end table`.

Setting up the table

Now you want to list the names of the sound files in the table. You could type all the names in by hand, but that can get tedious when you have lots more stimulus files to choose from. A faster way is to highlight all the sound files in the Finder window and then do a 'copy'. This will make the Finder put the names of the sound files onto the clipboard. Then you can just paste them into your script. After a little tidying-up you should have something like

```
table sounds
  Jaguar.wav
  Lion.wav
  Leopard.wav
end table
```

and that's all there is to your table.

Getting the value from the table

Now you have to tell the script that instead of using the one file you chose earlier

```
play sound Lion.wav
```

it should get the name of the sound file from the table. The simplest way to do this would be to replace that one line with this:

```
repeat using $varS from 1 row of sounds
  play sound $varS
end repeat
```

make that change, save your changes, and run the script.

This time you'll hear a different sound: the Jaguar one, because the Jaguar sound is listed first in the table (or at least, it is in mine). The instruction

```
repeat using $varS from 1 row of sounds
```

told PsyScript to pick 1 row of the sounds table and use the value it found there for the variable called `$varS`. Then the next line told it to play the sound in the file with the name in $varS. You have 26 variables to play with in PsyScript: `$varA` to `$varZ`. I chose the 'S' one because 's' stands for 'sound'.

You now have the question of how to get the other two sounds. Well, PsyScript knows that you want to have an equal chance of getting each sound, so it'll try to help. If, once one subject is run, you choose the 'run again' radio button it'll make sure that the next subject you run will get the next sound. However, if you stop running subjects and go back to looking at the script, it'll forget which rows of the table have been used and start again the next time you run the script.

Random order

So now you know how to get PsyScript to go all the way through the table. But it's still using the rows of the table in the order you typed them in. In order to avoid priming bias you might want PsyScript to pick a random order to present the stimuli in. To do this, you modify the table itself by adding `in random order` to the end of the `table` line:

```
table sounds in random order
```

If you make this change and run three subjects, one after the other, using the 'run again' radio button, you should get each of the three sounds in random order.

Using more than one table row in the same experiment

Well, we now have the script choosing randomly from the table for each subject, but usually you want each subject to encounter each stimulus, just in a different random order. The change to the `repeat using` line here is fairly simple: you just remove the bit that says `1 row of`:

```
repeat using $varS from 1 row of sounds
```

changes to

```
repeat using $varS from sounds
```

This is exactly equivalent to

```
repeat using $varS from all rows of sounds
```

So using all rows of a table is the default.

However, there's a problem with the script: the positioning of the repeat using and end repeat lines worked fine for this little demonstration, but they're not right for the bigger picture. The only bit of the script that's inside the repeat loop is the bit that plays the sound. So if you make the above change (go ahead and make it) you'll still get just one trial, but that one trial will play all three sounds (try it if you like).

The solution is to move the `repeat using` and `end repeat` lines. They belong at the beginning and end of the trial section of the script. The `repeat using` line belongs here:

```
show cell A
repeat using $varS from sounds
  wait for 5 seconds
```

(don't forget to delete it from where it was originally). The `end repeat` one belongs here:

```
    log $lastKey
    log $lastKeyTime
    log return
end repeat

display text from file thankyou.txt
```

(again, don't forget to remove the original). You can use the 'Check Syntax' button to make PsyScript get the indenting right for you.

If you run this version, you'll get one trial with each sound, which is just what you wanted. But if you look at the grid showing the results you'll see that although you can see the details of the response you can't tell which sound they're responding to. So just before the commands that log the response, log the sound that played:

```
log $varS
```

Run this and look at the log and you'll see you can now tell which stimulus triggered which response.

So now you have a different sound being played each trial, and you can tell which response goes with which sound.

Picture too

The next thing to do is to make PsyScript pick a different picture for each trial too. This could be done a number of ways. One would be to introduce another table for the pictures, and list the pictures there. But that way it would be difficult to ensure the correct combinations of pictures and sounds so instead you're going to expand the existing table by adding another column to it.

First, change the table's name from `sounds` to `stimuli`. Its name is used in two places: one's where the table itself is and the other is in the `repeat using` line that refers to it, so change both. Next add a new column to the end of the table so it looks like this:

```
table stimuli in random order
  Jaguar.wav,beast red.png
  Lion.wav,beast big.png
  Leopard.wav,beast normal.png
end table
```

you can separate the columns with commas or tabs. I've chosen commas. You can see from this that the red-eyed picture will always go with the sound of the Jaguar, the big-eyed picture will go with the sound of the lion, and the normal-eyed picture will go with the Leopard's growl.

Don't put anything extra in the table columns: no space after the commas, for instance.

That's all you have to do to the table itself. Now, we just have two changes to make to the procedure to make it get the name of the picture out of the table instead of always using the 'beast normal.png' picture. First, change the existing `repeat using` line

```
repeat using $varS from stimuli
```

to

```
repeat using $varS,$varP from stimuli
```

because the table went from having one value per row to two values per row. Now the script loads the first one into `$varS` and the second into `$varP`. I chose `$varP` because the second value is the name of a picture file.

The next bit is to change the line which currently always displays the same picture

```
load image beast normal.png into cell A
```

to

```
load image $varP into cell A
```

so it displays the picture whose filename has been loaded into `$varP`.

Lastly, while we're currently only logging which sound file was used, you should now also be logging which picture file was used too. Add the line

```
log $varP
```

where you think it will do most good. Save your script and run it.

Working the table

Since most of the work of providing stimuli is now done with the table, there's plenty you can do with the experiment without changing the script at all. For instance, instead of associating one sound with one picture you could provide all the possible combinations of them both to the subject merely by changing the table:

```
table stimuli in random order
  Jaguar.wav,beast red.png
  Jaguar.wav,beast big.png
  Jaguar.wav,beast normal.png
  Lion.wav,beast red.png
  Lion.wav,beast big.png
  Lion.wav,beast normal.png
  Leopard.wav,beast red.png
  Leopard.wav,beast big.png
  Leopard.wav,beast normal.png
end table
```

The table is still set to provide its data in random order so the subject will now see all the possible combinations in a random order. All done without changing a single command of your script.

Further

That's the end of this tutorial. Which is the last tutorial. So it's the end of all the tutorials.

You could develop this experiment further. In fact, you could reinterpret the experiment: perhaps the real experiment doesn't have anything to do with the eyes or growls, it's actually to do with whether they take a drink of water at the end of the experiment. The experimenter watches the subject during the experiment and if they drink from the cup before the end the experimenter notes it in the 'Comments' field. They also note whether the lights in the room were on or off during the experiment and these two factors are correlated.

Or you could do a different experiment by having two different pieces of text for two different sets of subject: one has a 'rateme.pict' file which says 'rate me' like the one I supplied. The other subjects see a picture which says 'rate this animal'. Maybe the 'rate me' form will encourage people to be more sympathetic to the animal and rate it as more scary.

And finally, you could take a look at the sample scripts supplied with PsyScript. They exploit many commands not explained in these tutorials and you should be able to write scripts for many experiments by looking at the techniques they use. Good luck.