In the today's lab, we will continue using MATLAB to program a visual search experiment of the sort you ran with *CogLab*.

In *Lab 7*, you created a function **FeatureSearchTrial** by following these steps:

1. Open a new **function** file in the MATLAB editor.

2. Put the function syntax line at the top of the file right after the word **function**:

[ TargetSeen, ResponseTime ] = **FeatureSearchTrial**( N_Distract, TargetPresent )

3. Before you do anything else, write the top comments that give the computational theory for your function. This will essentially be a summary of the description of this function on the preceding page. This should not, however, include details of implementation.

4. Now write out the main steps of your algorithm. I suggest:

% Set up the figure and triangle template.

*Hint: Use the functions* **SetUpFig** *and* **RedTriangle** *to do this.*

% Make *N_Distract* copies of the distractor triangle and
% place them at random integer coordinates within the figure.

*Hint: Use a* **for** *loop and* **CopyMovePatch**.
*You'll also need a HandleArray to store the handles for each new distractor you create. Initialize this before the loop using the* ***zeros*** *function.*

% If TargetPresent == true, change the color of the triangle template to yellow,
% and then move it to a random integer coordinate within the figure.
% Otherwise, make the triangle template invisible.

*Hint: Use the* **set** *function to change properties of the triangle, and use your* **MovePatch** *function to move it.*

% Record the response and response time.

*Hint: Use the function* **GetResponse** *to do this.*

5. Implement the algorithm using MATLAB code.
I recommend following the hints above.
TEST each section of code as you write it – don't just write the whole function first. You're more likely to be successful and besides, it's more fun that way!

If you still need to finish **FeatureSearchTrial**, do that now!

How to Avoid Placing Distractors and Targets on Top of Each Other

You may have noticed that sometimes when you run **FeatureSearchTrial** you see fewer distractors than you requested. This is because two or more distractors and/or the target have, randomly, been copied to the same location.

A good solution to this problem is the following.

Create a 100-by-2 array called *Locations* of all the possible (Delta_X, Delta_Y) pairs of integers between 0 and 9. You can do this with two nested **for** loops.

Then create a random selection vector *Select* using

Select = **randperm**(100);

The MATLAB function **randperm**(*n*) creates a random permutation of the vector 1:*n* that we will use for choosing distractor (and target) locations by selecting rows in *Locations*.

To make use of this approach RATHER THAN **randi** to choose *Delta_X* and *Delta_Y*, in the main **for** loop of **FeatureSearchTrial:**

**for** k = 1 : N_Distract
…
**end**

you will replace the lines involving **randi** with these lines:

```
DeltaX = Locations(Select(k), 1);
DeltaY = Locations(Select(k), 2);
```

Then, if *TargetPresent* == **true**, at some point after this **for** loop use

```
k = N_Distract+1
DeltaX = Locations(Select(k), 1);
DeltaY = Locations(Select(k), 2);
MovePatch(hrt, DeltaX, DeltaY);
```

These lines should replace your existing lines that use **randi** for locating the target.

Programming a Conjunction Search Display

The classic conjunction display has a target defined by the conjunction of two features, each of which characterizes one of two classes of distractors.

We currently have:

- red triangles as distractors, and
- a yellow triangle as the target.

Accordingly, we need a different yellow shape as the second class of distractor.

I suggest using a yellow diamond.

Modify the **RedTriangle** function to create a **YellowDiamond** function.

Use the following coordinates in setting up your template:

```
X = [ 0.5    1    0.5     0];
Y = [ 0     0.5    1    0.5];
```

Now …

Modify your revised **FeatureSearchTrial** function to create a **ConjunctionSearchTrial** function.

It should have equal numbers of both types of distractors if *N_Distract* is even. (That is, there should be *N_Distract*/2 distractors of each type.)

If *N_Distract* is odd, there will be at least (*N_Distract* – 1) / 2 distractors of each type. Then use **randi** to choose at random which type will get the extra distractor.

*Note*: I'm gradually giving you more freedom (fewer specific instructions) in how to write your programs. You should continue to use good programming principles in your work: writing out the top comments as a computational theory, writing out main section comments as an outline of your algorithm, choosing variable names that make sense in English, and testing as you go.

Save all this work for the next lab.