1. Write a function that gets the images, thresholds them, and creates a feature vector for each image. E.g., the easiest thing to do is to let each image be the feature vector:

```
  * Get image (say g1.jpg)   im = imread(g1.jpg)  or im = G(1).im
  * Normalize size: im = imresize(im,[15,15]);  for example
  * Threshold to make 0/1 image im = im>50;   - 50 is the gray level threshold
  * Now, add im to the samples:  X(1,:) = im(:);   this might need to be transposed.
  * might as well set the target vector, too:    y(1) = 1;     % where 1 will mean gold
  * Do this for each sample (there are 27) to produce:
    + X which is a 27x225 array (in this example)
    + y which is a 27x1 vector
```

2. You also need a perceptron learning function as described in the A9.pdf.

3. Next, set up a learning scenario (e.g., leave one out) where you learn a perceptron to classify gold (vs. pits and Wumpus). This is done by running the leave-one out method and averaging the error for each subset. I have attached a trace of one perceptron learning session (i.e., on one set of training data). You should get results like that for each perceptron learning run.

4. Once you have a gold classifier, then find a pit classifier perceptron.

5. Run these on all the data to show you how well it is classified in the final result.

Then try the process again using some other image features. One would be to compute the magnitude and orientation of edges in the image:

```
[dx,dy] = gradient(im);
mag = sqrt(dx.^2+dy.^2);
ori = atan2(dy,dx);
```

Then pull out all the orientations for pixels where the magnitude is large enough, and write a histogram function which bins the orientations into say 10 bins, and use that as the feature vector for each image. I.e., X will now be a 27x10 array (y remains a 27x1 vector).