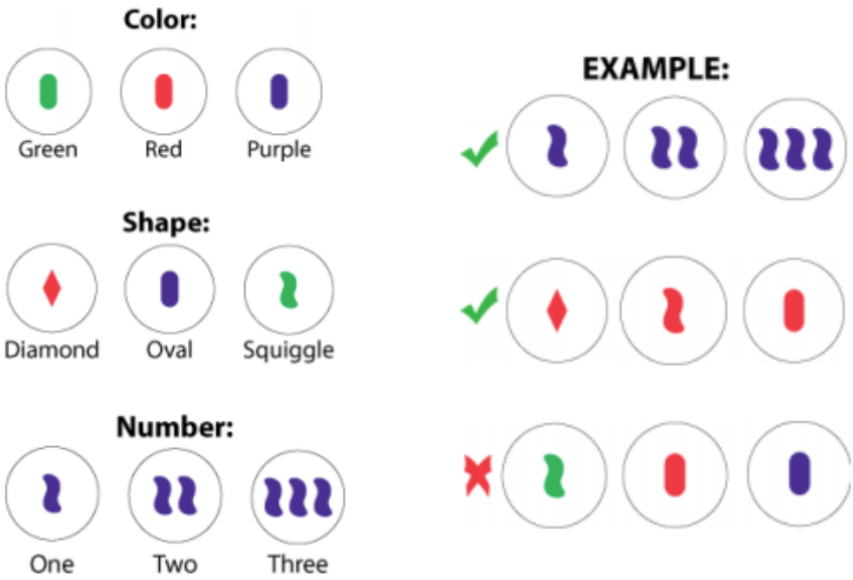
A program to play “Set”

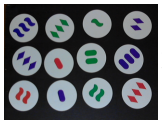


Justin Pearson
Sep 21, 2017

We use Mathematica's image-processing functions and a little machine learning to play the pattern-matching card game "Set".
For more information see:
http://justinppearson.com/presentations.html#a-program-to-play-the-set-card-game
https://www.setgame.com/sites/default/files/instructions/SET%20Mini%20Round%20Instructions.pdf

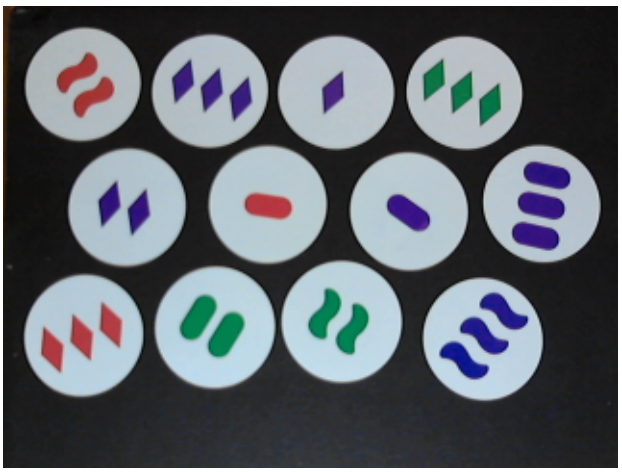
In[1]:=
```
Clear["Global`*"]
SetDirectory[NotebookDirectory[]];
```

Enable / disable webcam usage (in case no webcam)

In[3]:=
```
USEWEBCAM = False;
If[USEWEBCAM,
 Print[$ImagingDevices];
 $ImagingDevice = "Logitech Camera";
 currentImage[] = CurrentImage[];
 ,
 currentImage[] =  ;
]
```

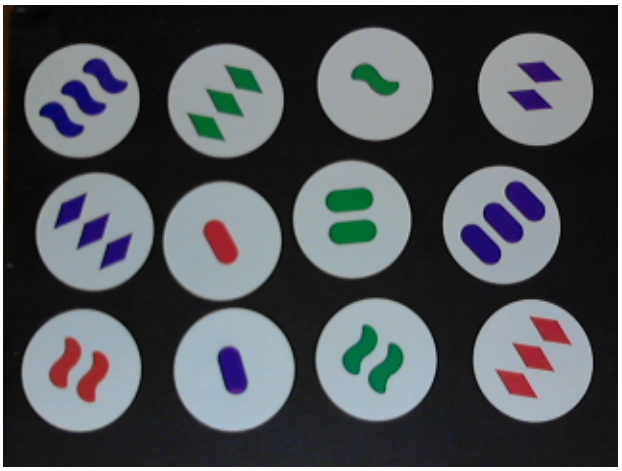
Camera calibration (do this manually)

Adjust the lighting to make the CurrentImage[] match these pictures:



In[5]:=
```
currentImage[]
```

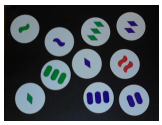
Out[5]=



Basic definitions

In[6]:=
```
COUNTS = {1, 2, 3};
COLORS = {Darker@Red, Darker@Darker@Blue, Darker@Darker@Green};
SHAPES = {"diamond", "oval", "squiggly"};
SHAPEATTRIBUTES = {"Rectangularity",
    "BoundingDiskCoverage", "ConvexCoverage", "FilledCircularity"};
```

Main algorithm

In[10]:=
```
f[frame_:
```

```
, debug_: False, quitOnFirstSet_: False] := Module[
  {start, frameAdj, mask, cards, cardImsAndBoxes, i, cardIm, bbox, cardmask,
   blobdata, cardCount, blobColors, blobShapes, j, d, A, x, cardColor,
   cardShape, newCard, boundingBoxes, pairs, k, triplet, RESULTS},

  RESULTS = {};
  AppendTo[RESULTS, DateString[{"DayName", ", ", "MonthName", " ", "Day", " ",
     "Year", " ", "Hour12", ":", "Minute", ":", "Second", ".", "Millisecond"}]];
  start = Now;
  frameAdj = frame ; (*ImageAdjust[frame,{.2,.6}]*)
  mask = frameAdj // Binarize // FillingTransform // Erosion[#, 1] &;
  cards = {};
  cardImsAndBoxes =
   Values@ComponentMeasurements[{frameAdj, MorphologicalComponents[mask]},
     {"MaskedImage", "BoundingBox"}, #Area > 50 && #FilledCircularity > .94 &];
```

```
(* For each card, parse it. *)

For[i = 1, i ≤ Length@cardImsAndBoxes, i++,
 {cardIm, bbox} = cardImsAndBoxes[[i]];
 cardmask = MorphologicalBinarize[cardIm,
    FindThreshold[cardIm, Method → "Mean"]] // ColorNegate;
 blobdata = Values@ComponentMeasurements[
    {cardIm, cardmask // MorphologicalComponents},
    {"MaskedImage", "Median"} ~Join~ SHAPEATTRIBUTES,
    #Area > 20 && #AdjacentBorderCount == 0 &, "ComponentPropertyAssociation"];
 cardCount = Length@blobdata;
 blobColors = {};
 blobShapes = {};

 (* For each blob on the card, classify its color and shape. *)

 For[j = 1, j ≤ Length@blobdata, j++,
  d = blobdata[[j]];
  AppendTo[blobColors,
   MinimalBy[COLORS, ColorDistance[RGBColor[d["Median"]], #] &] // First];
  (* These numbers come from the shape classifier
   developed in a later section: *)
  A = {{-121.329, -251.667, -49.961, 100.653, 244.074},
       {-361.618, 100.242, -76.6762, -2.51293, 399.766},
       {0., 0., 0., 0., 0.}};
  x = Flatten@{1, Values@d[SHAPEATTRIBUTES]};
  AppendTo[blobShapes, SHAPES[[First@Ordering[A.x, -1]]]];
 ];

 (* Show me the card you've parsed. *)

 cardColor = Commonest[blobColors, 1] // First;
 cardShape = Commonest[blobShapes, 1] // First;
 newCard = {cardCount, cardColor, cardShape};
 If[debug, AppendTo[RESULTS,
    Framed@Row[#, " "] &@{
      cardIm, cardmask,
      Column@{Row[newCard, " "],
        Grid[Transpose@{blobdata[[All, "MaskedImage"]], blobColors, blobShapes},
         Frame → True]}}];
 ];
 boundingBoxes[newCard] = bbox;
```

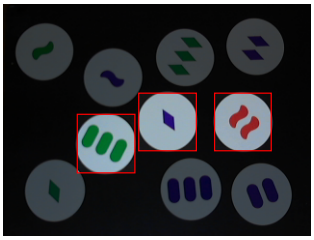
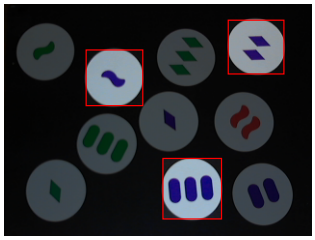
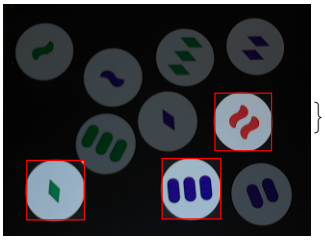
```
  (* For each new triplet, tell me if it's a set. *)

  If[Length@cards ≥ 2,
   pairs = Subsets[cards, {2}];
   For[k = 1, k ≤ Length@pairs, k++,
    triplet = Join[pairs⟦k⟧, {newCard}];
    If[triplet // Transpose // Map[Counts] // Map[Length] // ContainsOnly[{1, 3}],
     AppendTo[RESULTS, HighlightImage[frame,
       Rectangle @@@ boundingBoxes /@ triplet, {"Darken", .6}]];
     If[quitOnFirstSet, Return[RESULTS]];
    ];
   ];
  ];
  AppendTo[cards, newCard];
 ];
 Return[RESULTS]
]
```

Examples

No args: find sets in an example image:

In[11]:=
```
f[]
```

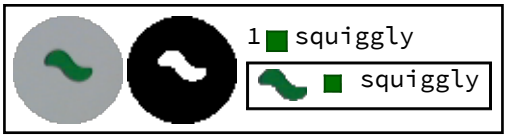
Out[11]=
```
{Sunday, January 14 2018 01:55:36.505,
```


,

,

}

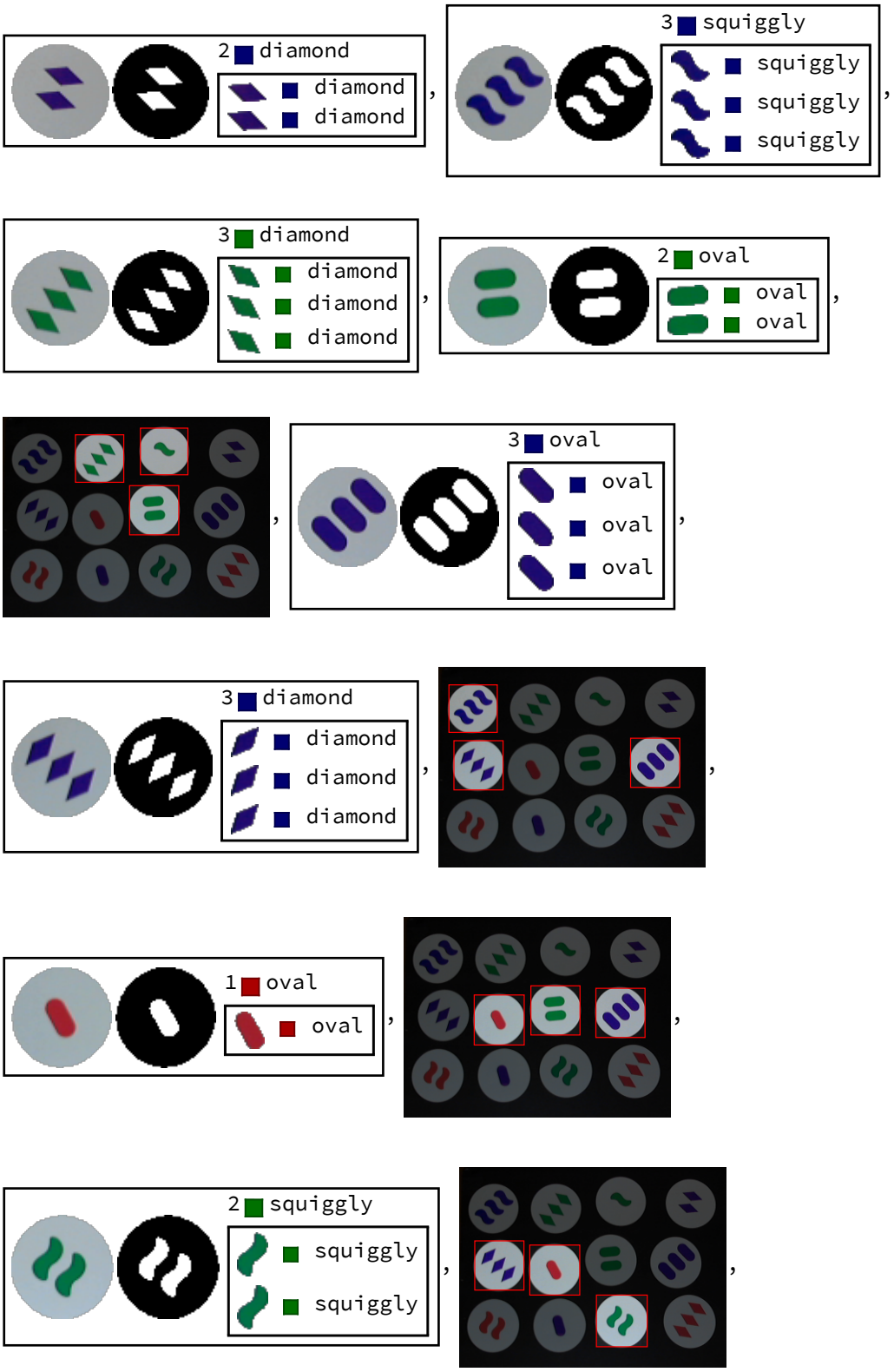
2nd arg: “debug” (show cards as you parse them):

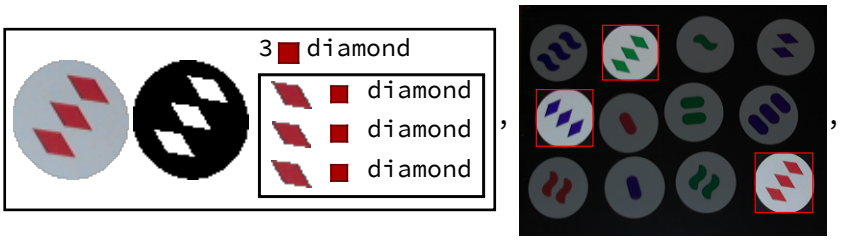
In[12]:=
```
f[currentImage[], True]
```

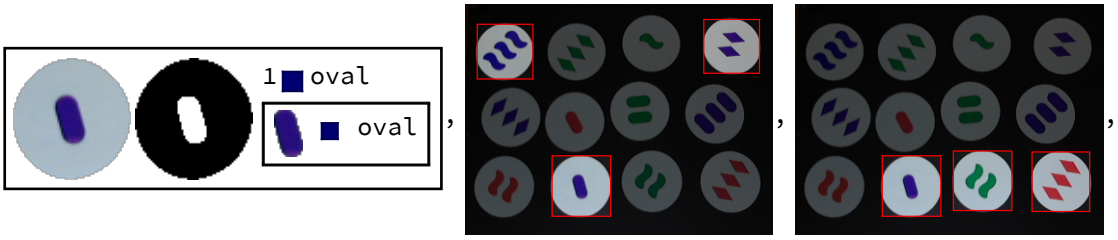
Out[12]=
```
{Sunday, January 14 2018 01:55:36.944,
```

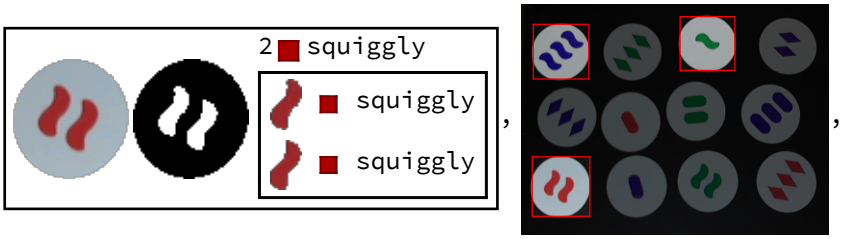


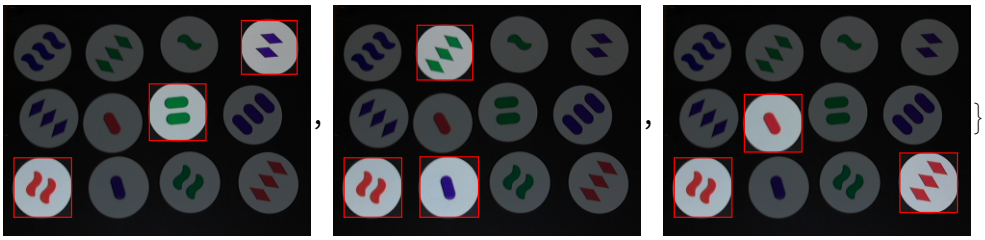
,


2 diamond
diamond
diamond
3 squiggly
squiggly
squiggly
squiggly
3 diamond
diamond
diamond
diamond
2 oval
oval
oval
3 oval
oval
oval
oval
3 diamond
diamond
diamond
diamond
1 oval
oval
2 squiggly
squiggly
squiggly





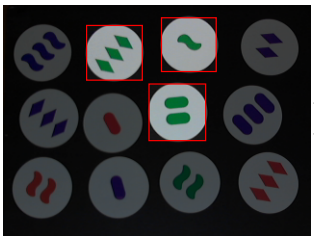




3rd arg: “quit after finding 1st set”:

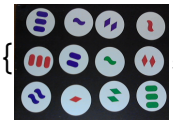
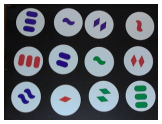
In[13]:=
```
f[currentImage[], False, True]
```

Out[13]= {Sunday, January 14 2018 01:55:38.129,

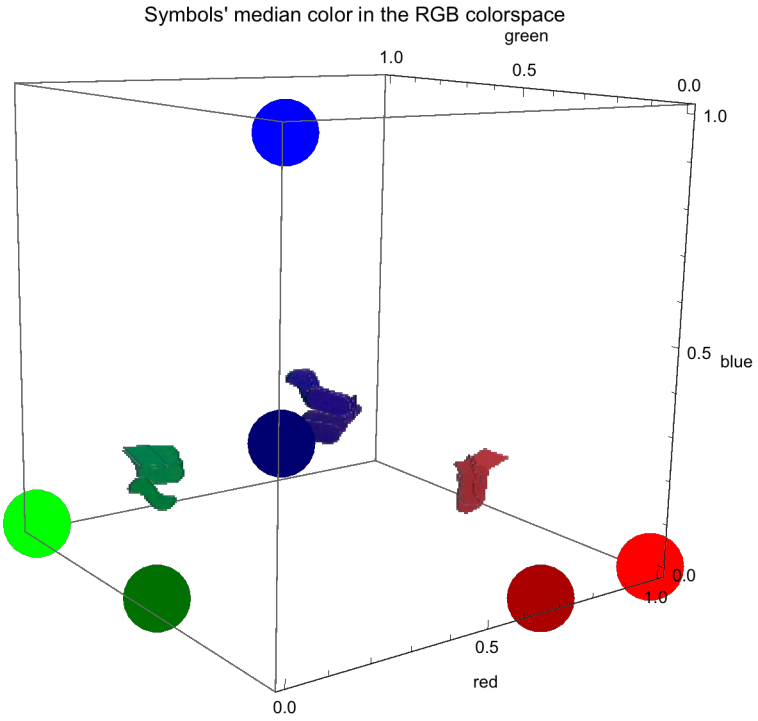

}

Color classifier

Simple “nearest neighbor” algorithm: blob is closest to which color?

In[14]:=
```
Table[
  Inset[blob["MaskedImage"], Most@blob["Median"]],

  {card, Values@

    ComponentMeasurements[{
```

```
, 
```

```
// Binarize // FillingTransform //

      Erosion[#, 1] & // MorphologicalComponents},
    "MaskedImage", #Area > 50 && #FilledCircularity > .94 &]},

  {blob, Values@ComponentMeasurements[
      {card, MorphologicalBinarize[card, FindThreshold[card, Method → "Mean"]] //
        ColorNegate // MorphologicalComponents},
      {"MaskedImage", "Median"},
      #Area > 20 && #AdjacentBorderCount == 0 &, "ComponentPropertyAssociation"] //
    Flatten[#, 1] &}
] //
Graphics3D[{#,
   Table[{PointSize[.1], c, Point[List @@ c]},
    {c, Union@Join[COLORS, {Red, Green, Blue}]}]},
  Axes → True,
  AxesLabel → {"red", "green", "blue"},
  PlotLabel → "Symbols' median color in the RGB colorspace"] &
```

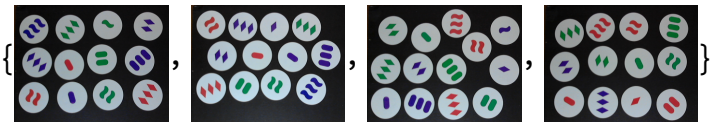
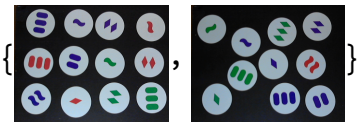
Out[14]=



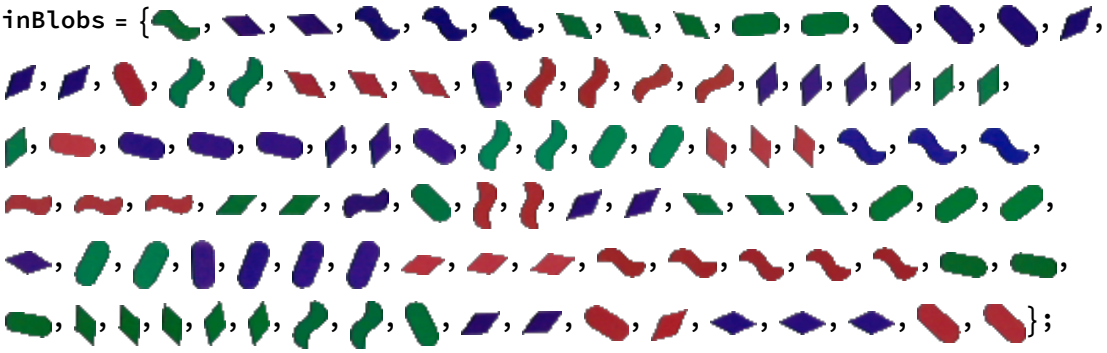
Shape classifier

We use Multinomial Logistic Regression using 4 shape properties, then train with Maximum-Likelihood Estimation of the decision boundaries.

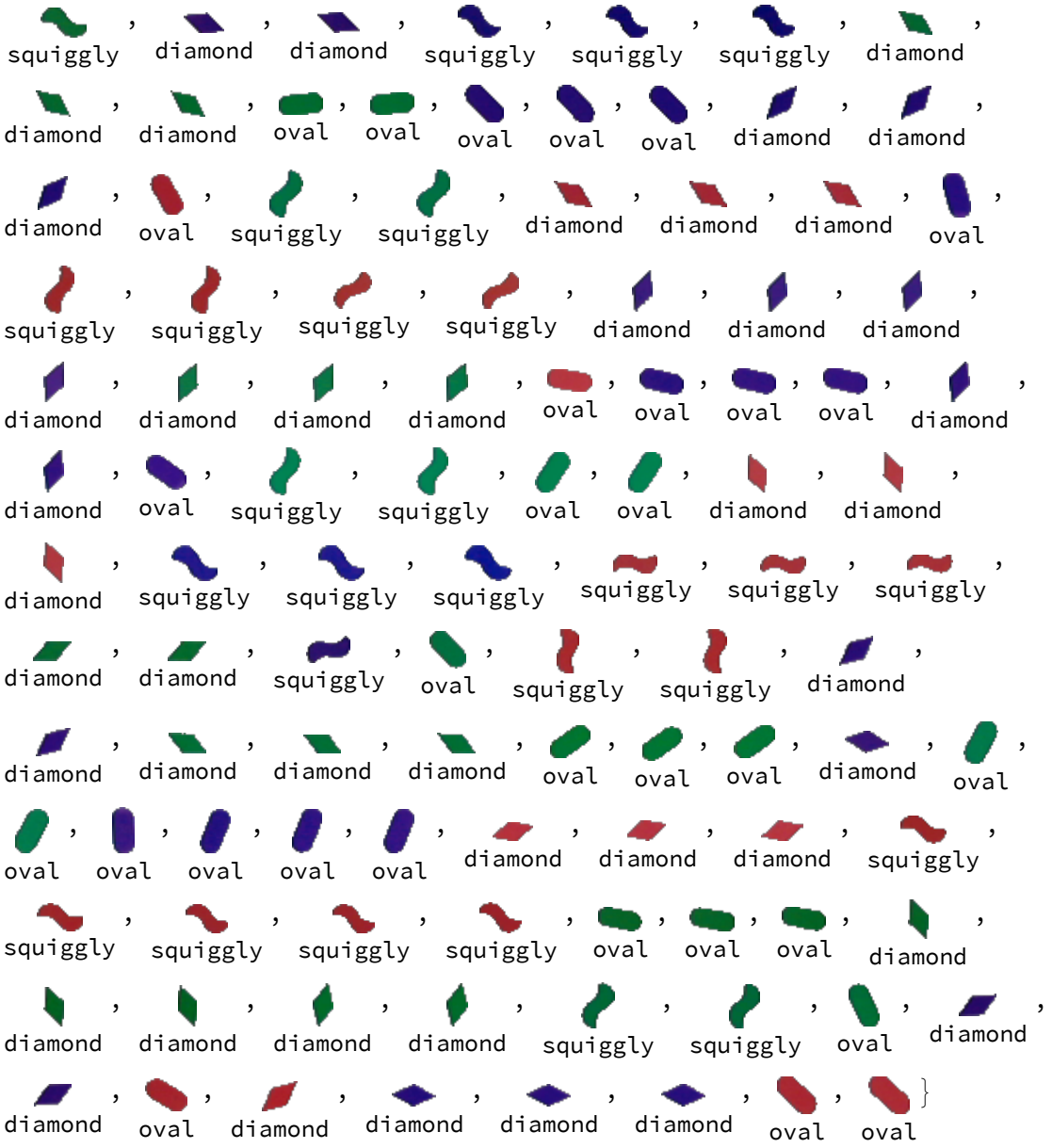
Gather & Prep Training Data / Test Data

In[15]:=
```
trainingFrames = {
```

```
};
testFrames = {
```

```
};
```

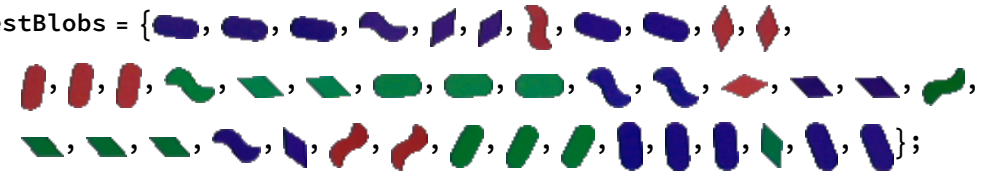
“Training” frames:

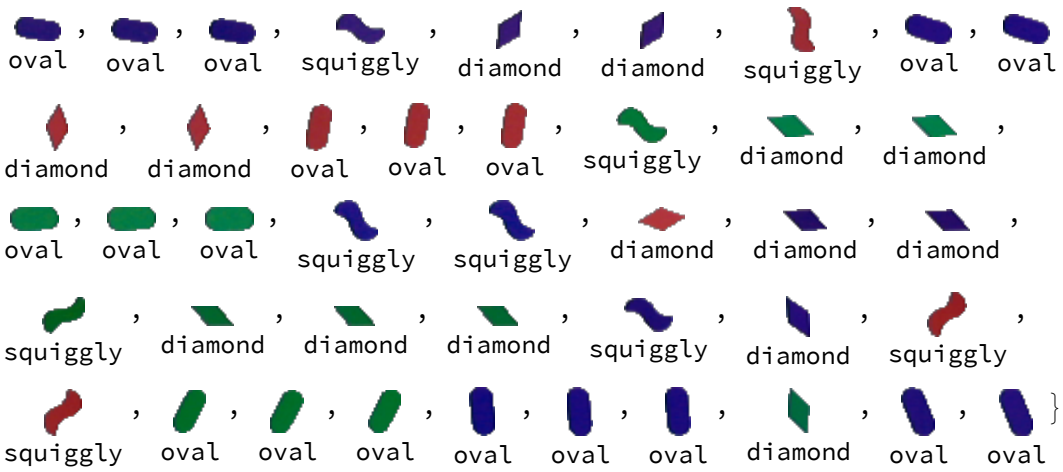
In[17]:=
```
trainBlobs = {
```

```
};
trainTrueLabels =
  ("sddssddddooooodddossddddossssddddddddooooddossoodddsssssddsossddddddooodooooood
      ddsssssooodddddssoddoddddoo" // Characters) /.
   {"o" → "oval", "d" → "diamond", "s" → "squiggly"};
MapThread[Labeled[#1, #2] &, {trainBlobs, trainTrueLabels}]
```

Out[19]=


squiggly diamond diamond squiggly squiggly squiggly diamond
diamond diamond oval oval oval oval oval diamond diamond
diamond oval squiggly squiggly diamond diamond diamond oval
squiggly squiggly squiggly squiggly diamond diamond diamond
diamond diamond diamond diamond oval oval oval oval diamond
diamond oval squiggly squiggly oval oval diamond diamond
diamond squiggly squiggly squiggly squiggly squiggly squiggly
diamond diamond squiggly oval squiggly squiggly diamond
diamond diamond diamond diamond oval oval oval diamond oval
oval oval oval oval oval diamond diamond diamond squiggly
squiggly squiggly squiggly squiggly oval oval oval diamond
diamond diamond diamond diamond squiggly squiggly oval diamond
diamond oval diamond diamond diamond diamond oval oval

“Testing” frames:

In[20]:=
```
testBlobs = {  };
testTrueLabels = ("ooosddsooddooosddooossdddsdddsdssooooooodoo" // Characters) /.
   {"o" → "oval", "d" → "diamond", "s" → "squiggly"};
MapThread[Labeled[#1, #2] &, {testBlobs, testTrueLabels}]
```

Out[22]=


Gather feature vectors

In[23]:=
```
{trainFeatureVecs, testFeatureVecs} = Table[
   ComponentMeasurements[blob, SHAPEATTRIBUTES] // Values // Flatten[#, 1] &,
   {blobs, {trainBlobs, testBlobs}},
   {blob, blobs}
  ];
```

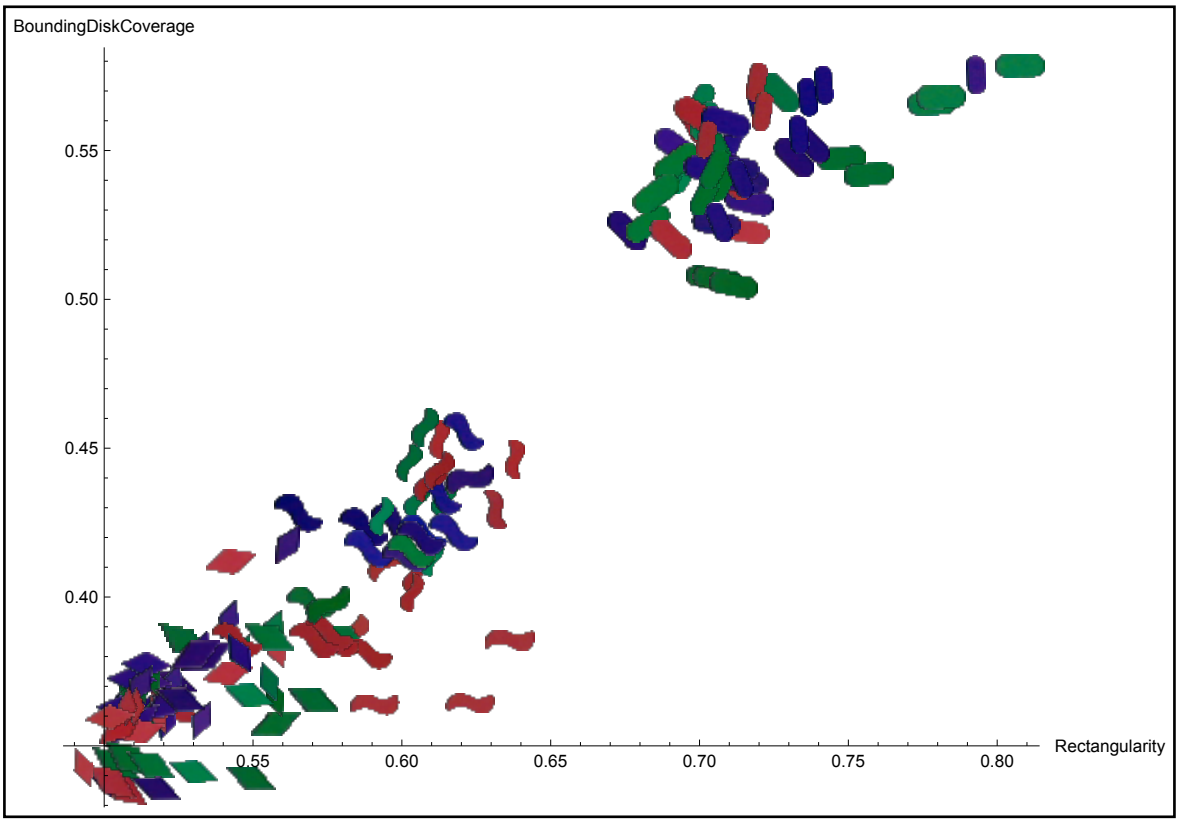
In[24]:=
```
Length /@ {trainFeatureVecs, testFeatureVecs}
```

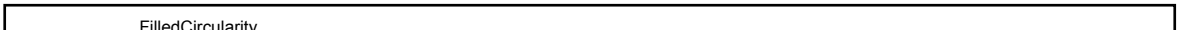
Out[24]=
```
{104, 42}
```

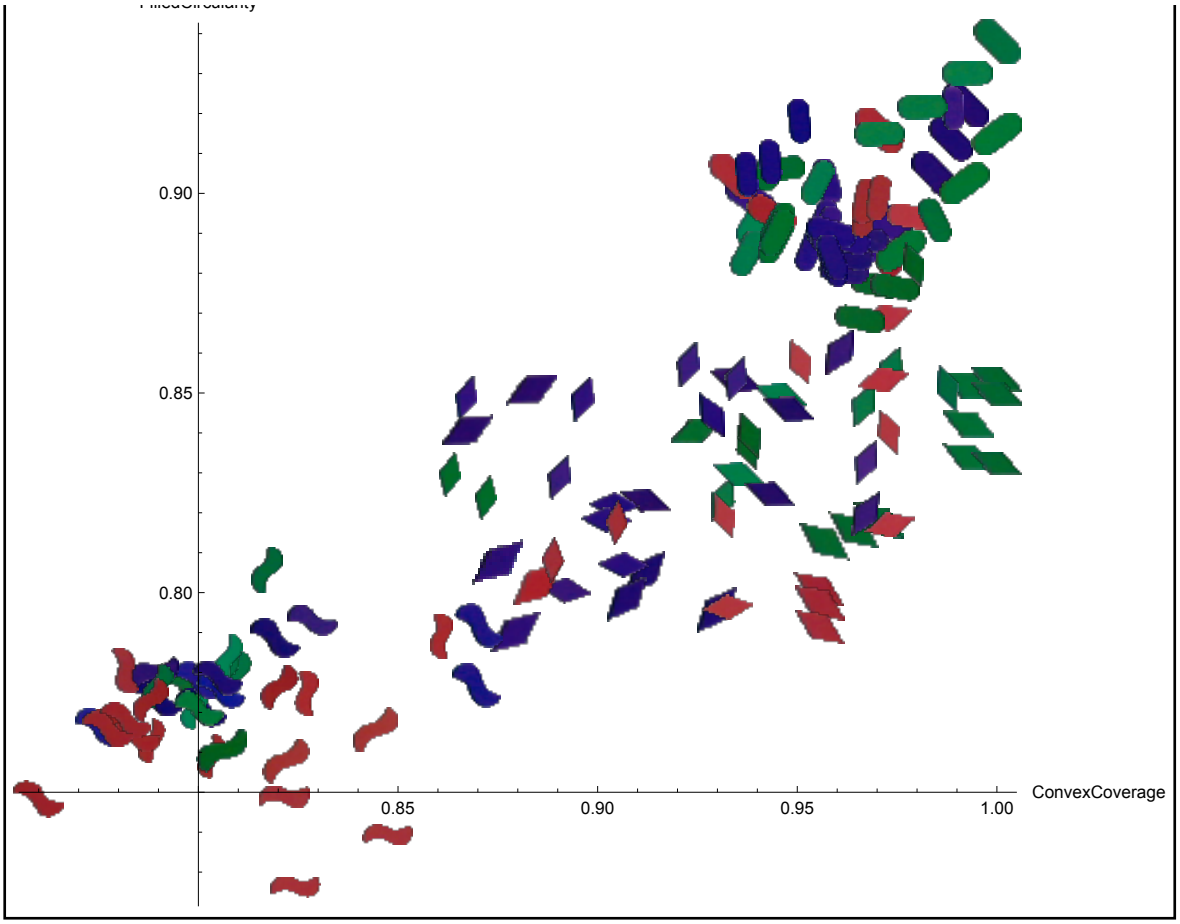
Plot training and test sets

In[25]:=
```
Table[
    Graphics[
     Table[
      Inset[blob["MaskedImage"], blob /@ attrs, Automatic, Scaled[.05]],
      {frame, Join[trainingFrames, testFrames]},
      {card,
       Values@ComponentMeasurements[{frame, frame // Binarize // FillingTransform //
            Erosion[#, 1] & // MorphologicalComponents},
         "MaskedImage", #Area > 50 && #FilledCircularity > .94 &]},
      {blob, Values@ComponentMeasurements[
          {card, MorphologicalBinarize[card, FindThreshold[card, Method → "Mean"]] //
             ColorNegate // MorphologicalComponents},
          Join[{"MaskedImage"}, attrs],
          #Area > 20 && #AdjacentBorderCount == 0 &,
          "ComponentPropertyAssociation"] // Flatten[#, 1] &}
     ], Axes → True, AxesLabel → attrs, ImageSize → 600]
    , {attrs, Partition[SHAPEATTRIBUTES, 2]}
   ] // Map[Framed] // Row
```

Out[25]=







Train classifier

In[26]:=
```
X = trainFeatureVecs;
Short[X, 5]
```

Out[27]//Short=
```
{{0.57409, 0.405399, 0.792829, 0.77498}, {0.5228, 0.371578, 0.934641, 0.853573},
 {0.506944, 0.339221, 0.918239, 0.825781}, <<98>> ,
 {0.51402, 0.377167, 0.902299, 0.81824}, {0.710664, 0.539854, 0.970696, 0.915806},
 {0.690231, 0.520165, 0.933798, 0.904164}}
```

In[28]:=
```
Y = trainTrueLabels;
Short[Y]
```

Out[29]//Short=
```
{squiggly, diamond, diamond, squiggly, squiggly, squiggly,
 diamond, <<90>> , oval, diamond, diamond, diamond, diamond, oval, oval}
```

In[30]:=
```
LABELS=SHAPES;
```

In[31]:=
```
k = Length@LABELS
{m, n} = Dimensions@X
```

Out[31]= 3

Out[32]= {104, 4}

In[33]:=
```
X = X // Map[Prepend[1]]; (* for the hyperplanes' offsets;
see Andrew Ng's notes cs229_notes1.pdf *)
vars = Array[θ, {k, n + 1}];
constraint = # == 0 & /@ Last[vars];
(* Reduce redundancy (all planes could be shifted w/o chaning the classifier;
  to prevent, constrain one plane to be just f(x)=0; see Ng's notes. *)
```

In[36]:=
```
loglikelihood[θ_] :=
```

$$\sum_{i=1}^{m}\left(\text{Log}@\prod_{l=1}^{k}\left(\frac{e^{\theta[\![l]\!].X[\![i]\!]}}{\sum_{j=1}^{k} e^{\theta[\![j]\!].X[\![i]\!]}}\right)^{\text{Boole}[Y[\![i]\!]==\text{LABELS}[\![l]\!]]}\right)$$

In[37]:=
```
AbsoluteTiming[
 A = vars /. Last@FindMaximum[{loglikelihood[vars], constraint}, Flatten@vars]]
```

Out[37]= {7.67216, {{-245.376, -142.033, -225.693, 83.7461, 430.352},
  {-361.625, 31.9662, -95.7845, 55.5154, 403.845}, {0., 0., 0., 0., 0.}}}

Examples

In[38]:=
```
Table[{
    blob,
    x = ComponentMeasurements[blob, SHAPEATTRIBUTES] // Values // First // Prepend[1],
    A.x,
    LABELS[[First@Ordering[A.x, -1]]]
   }, {blob, {  ,  ,  }}] //
  TableForm[#,
    TableDepth → 2,
    TableAlignments → Center,
    TableHeadings →
     {None, {"shape", "x=rectangularity etc", "A*x", "classified shape"}}
   ] &
```

Out[38]//TableForm=

shape	x=rectangularity etc	A*x	clas
	{1, 0.705561, 0.525649, 0.961538, 0.88058}	{-4.74043, 19.5778, 0.}	
	{1, 0.51032, 0.3631, 0.904762, 0.817582}	{27.8106, 0.31323, 0.}	
	{1, 0.620569, 0.455728, 0.870229, 0.791842}	{-22.7232, -17.347, 0.}	

Validation:

In[39]:=
```
({im, v, true} ↦ Block[{pred},
    pred = LABELS[[First@Ordering[A.Prepend[v, 1], -1]]];
    {im, v, true, Style[pred, If[true == pred, Darker@Green, Red]]}]) @@@
  ({testBlobs, testFeatureVecs, testTrueLabels}ᵀ) //
 TableForm[#, TableHeadings → {None, {"image", "shape stats", "true", "predicted"}},
   TableDepth → 2, TableAlignments → Center] &
```

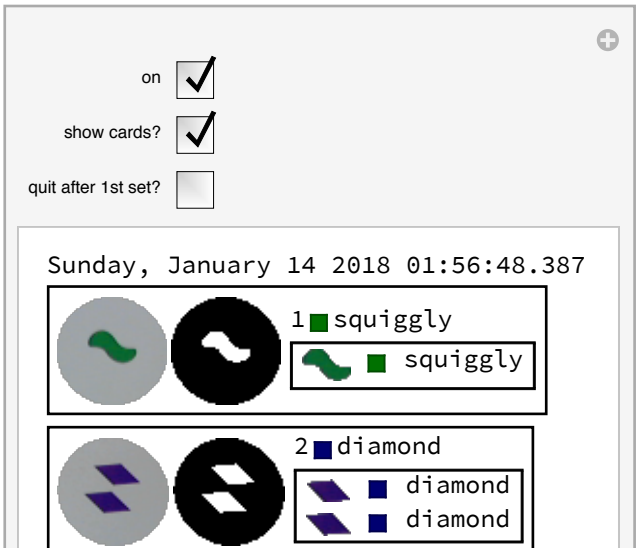
Out[39]//TableForm=

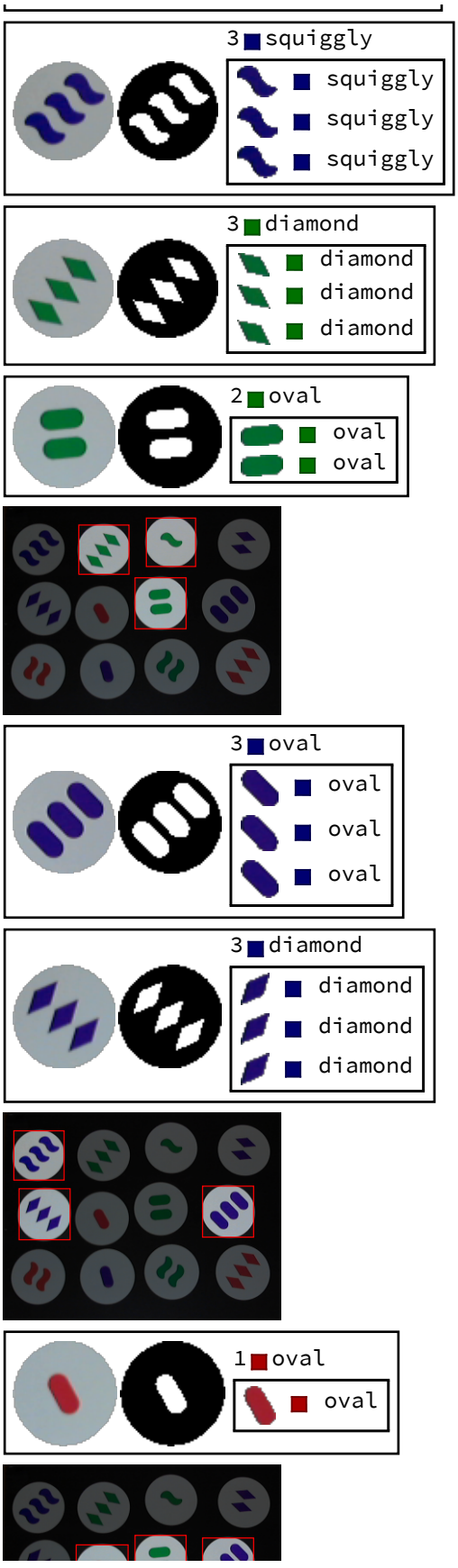
image	shape stats	true	predicted
	{0.705561, 0.525649, 0.961538, 0.88058}	oval	oval
	{0.714978, 0.539667, 0.966527, 0.888673}	oval	oval
	{0.711933, 0.544339, 0.96281, 0.883971}	oval	oval
	{0.601687, 0.413548, 0.828571, 0.79291}	squiggly	squiggly
	{0.515522, 0.373147, 0.961039, 0.813598}	diamond	diamond
	{0.561538, 0.417737, 0.960526, 0.860124}	diamond	diamond
	{0.631277, 0.429324, 0.781609, 0.780505}	squiggly	squiggly
	{0.702756, 0.54322, 0.961977, 0.888574}	oval	oval
	{0.708888, 0.55952, 0.958491, 0.890328}	oval	oval
	{0.51032, 0.3631, 0.904762, 0.817582}	diamond	diamond
	{0.508121, 0.3631, 0.888889, 0.808189}	diamond	diamond
	{0.70309, 0.551348, 0.963265, 0.889643}	oval	oval
	{0.718647, 0.575406, 0.967078, 0.899936}	oval	oval
	{0.724509, 0.565612, 0.970588, 0.900984}	oval	oval
	{0.603027, 0.415597, 0.8, 0.771144}	squiggly	squiggly
	{0.536667, 0.345686, 0.947059, 0.834969}	diamond	diamond
	{0.552083, 0.369425, 0.946429, 0.848786}	diamond	diamond
	{0.781065, 0.567796, 0.977778, 0.91468}	oval	oval
	{0.784024, 0.569947, 0.974265, 0.916411}	oval	oval
	{0.807692, 0.57836, 0.992727, 0.930141}	oval	oval
	{0.620569, 0.455728, 0.870229, 0.791842}	squiggly	squiggly
	{0.613078, 0.433972, 0.869732, 0.776643}	squiggly	squiggly
	{0.501897, 0.359726, 0.932515, 0.79651}	diamond	diamond
	{0.525362, 0.365584, 0.947712, 0.846014}	diamond	diamond
	{0.517361, 0.335775, 0.943038, 0.824905}	diamond	diamond
	{0.581068, 0.401325, 0.8107, 0.771076}	squiggly	squiggly
	{0.549407, 0.339043, 1., 0.832328}	diamond	diamond
	{0.570248, 0.365295, 1., 0.853363}	diamond	diamond
	{0.570248, 0.365295, 1., 0.849756}	diamond	diamond
	{0.592828, 0.425771, 0.803846, 0.784146}	squiggly	squiggly
	{0.541958, 0.378795, 0.922619, 0.841921}	diamond	diamond

	{0.60962, 0.438328, 0.820313, 0.776181}	squiggly	squiggly
	{0.612613, 0.442503, 0.788104, 0.773027}	squiggly	squiggly
	{0.707744, 0.539281, 0.945098, 0.891947}	oval	oval
	{0.70187, 0.534805, 0.944664, 0.888238}	oval	oval
	{0.704807, 0.543732, 0.944882, 0.890094}	oval	oval
	{0.736264, 0.567767, 0.950355, 0.918046}	oval	oval
	{0.733516, 0.555482, 0.943463, 0.907918}	oval	oval
	{0.741823, 0.572004, 0.9375, 0.904698}	oval	oval
	{0.555556, 0.371749, 0.987654, 0.851451}	diamond	diamond
	{0.713693, 0.540691, 0.958763, 0.882272}	oval	oval
	{0.708978, 0.527836, 0.962069, 0.885473}	oval	oval

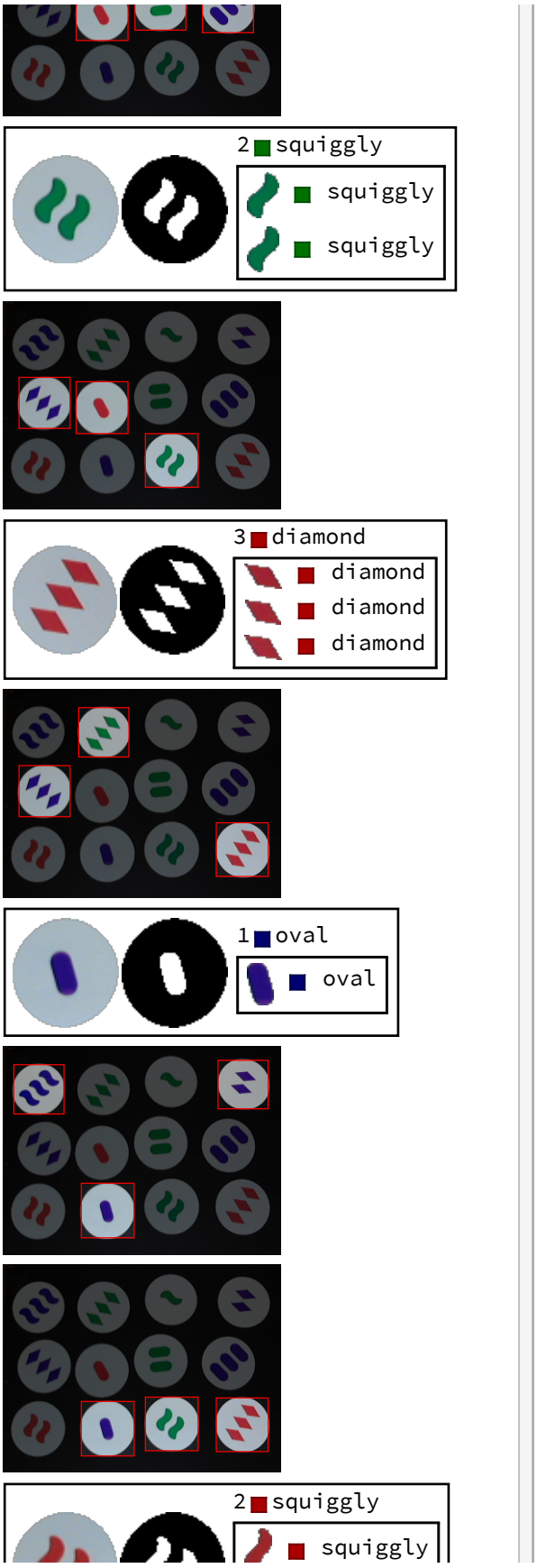
Play Set

In[40]:=
```
Manipulate[
 If[on,
  Column@f[currentImage[], debug, quitOnFirstSet],
  "Push \"on\" to start playing."
 ],
 {{on, True}, {True, False}},
 {{debug, True, "show cards?"}, {True, False}},
 {{quitOnFirstSet, False, "quit after 1st set?"}, {True, False}},
 SaveDefinitions → True
]
```




3 squiggly
squiggly
squiggly
squiggly
3 diamond
diamond
diamond
diamond
2 oval
oval
oval
3 oval
oval
oval
oval
3 diamond
diamond
diamond
diamond
1 oval
oval

Out[40]=


2 squiggly
squiggly
squiggly
3 diamond
diamond
diamond
diamond
1 oval
oval
2 squiggly
squiggly


squiggly