

连连看实验报告

郭一隆 (2013011189)

September 13, 2015

Contents

1 原创性	4
2 制作自己的连连看	4
3 攻克别人的连连看	14
4 实验总结	37

List of Figures

2.1 运行连连看	4
2.2 重写 <code>detect.m</code> 后运行 <code>linkgame</code>	8
2.3 测试 <code>matchadj()</code> 功能	9
2.4 定义上边界	10
2.5 依次调用 <code>matchadj</code> 和 <code>matchborder</code> 后	12
3.1 扫描线灰度均值	14
3.2 图像分割效果	16
3.3 <code>graycapture</code> 预处理成黑白图像	17
3.4 扫描线黑白均值	17
3.5 <code>findpeaks</code> 检测峰值 (纵坐标为原数据的相反数)	18
3.6 利用二值图像法得到的图像分割效果	19
3.7 高通滤波提取纹理特征	20
3.8 相似度走势曲线	21
3.9 相似度最高的十对图像块	22
3.10 不同块最相似前十名	24
3.11 不做高通滤波直接利用相似度匹配的误判前十名	24
3.12 对 <code>graycapture</code> 自动分类结果	27
3.13 模拟消除过程 (部分)	36

List of Tables

3.1 <code>finkpeaks</code> 函数参数设置	18
3.2 精灵种类对应表 (人脑识别)	23

List of Source Codes

2.1 <code>canlink.m(adjcross)</code>	5
2.2 <code>canlink.m(canlink0)</code>	5
2.3 <code>canlink.m(canlink1)</code>	6
2.4 <code>canlink.m(main)</code>	7
2.5 <code>detect.m(main)</code>	7
2.6 <code>omg.m(matchadj)</code>	9
2.7 <code>omg.m(matchborder)</code>	11
2.8 <code>omg.m(matchrest)</code>	13
2.9 <code>omg.m(main)</code>	13

3.1	<code>divide.m</code> 获取块大小以及位置信息	15
3.2	<code>divide.m</code> 获取所有块	15
3.3	<code>similarity.m</code>	20
3.4	<code>similar sort.m</code>	21
3.5	手动标定精灵种类	22
3.6	<code>classify.m</code>	26
3.7	<code>mocklink.m(1-22, get mtx)</code>	28
3.8	<code>findpath.m</code>	31
3.9	<code>solve linkgame.m</code>	34
3.10	<code>mocklink.m(22-74, visualization)</code>	35

1 原创性

本次连连看大作业由本人独立思考完成，如有雷同，纯属巧合。

2 制作自己的连连看

1. 在 MATLAB 环境下，设置当前路径为 linkgame，运行 linkgame(打开linkgame.fig或右键linkgame.p点“运行”)，熟悉游戏。

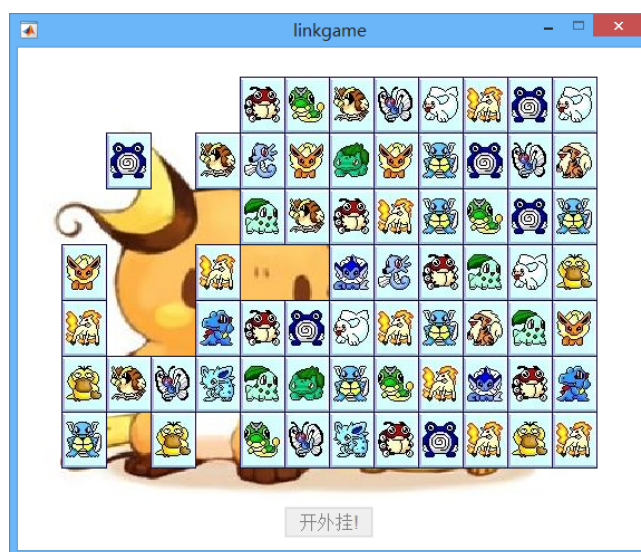


Figure 2.1: 运行连连看

2. 注意 linkgame 目录下有个 detect.p，它的功能是检测块是否可以消除。现将其删掉，然后把 linkgame\reference 目录下的detect.m复制到 linkgame 目录下。detect.m文件中是 detect 函数，函数以图像块的索引矩阵与要判断的两个块的下标为输入，如果两个块能消掉则输出 1，否则输出 0。根据文件中的注释提示，实现判断块是否可以消除的功能。写完后再次运行 linkgame，检验游戏是否仍然可以正确运行。

算法实现分为以下几个步骤：

- (a) 画十字 adjcross：在给定点周围空白处画十字，遇到块则停止。

```

29 function [I,J] = adjcross(mtx,x,y)
30 % return index vector of can-reach blocks of mtx(x,y) on both directions
31 % notice: x >= 2 && y >= 2 && all(mtx >= 0)
32
33 % get vectors of four directions
34 left = mtx(x,1:y-1);
35 right = mtx(x,y+1:end);
36 up = mtx(1:x-1,y);
37 down = mtx(x+1:end,y);
38
39 % get zero run length adjacent to mtx(x,y)
40 lz = find(cumsum(left,'reverse')==0);
41 rz = find(cumsum(right,'forward')==0);
42 uz = find(cumsum(up,'reverse')==0);
43 dz = find(cumsum(down,'forward')==0);
44
45 I = [x*ones(1,length([lz,rz])),uz.',x+dz.'];
46 J = [lz,y+rz,y*ones(1,length([uz,dz]))];
47
48 end

```

Listing 2.1: canlink.m(adjcross)

(b) 判断是否可直连 canlink0 : 先判断横纵坐标是否在同一直线, 再确认路径上是否有障碍。

```

51 function bool = canlink0(mtx,x1,y1,x2,y2)
52 % return 1 if it's a direct link (no turns)
53
54 if x1 == x2 && ~any(mtx(x1,min(y1,y2)+1:max(y1,y2)-1))
55     bool = 1;
56 elseif y1 == y2 && ~any(mtx(min(x1,x2)+1:max(x1,x2)-1,y1))
57     bool = 1;
58 else
59     bool = 0;
60 end
61
62 end

```

Listing 2.2: canlink.m(canlink0)

(c) 判断是否可用不超过一个直角的连线连接 canlink1 : 选取两目标点之一作为起点, 画十字; 对十字上的点进行遍历, 检查是否存在可与另一目标点直连的点。

```

65 function bool = canlink1(mtx,x1,y1,x2,y2)
66 % return 1 if the turns of link path <= 1
67
68     if canlink0(mtx,x1,y1,x2,y2)
69         bool = 1;
70         return
71     end
72
73     % grow the cross of origin
74     [I,J] = adjcross(mtx,x1,y1);
75
76     for n = 1:length(I)
77         i = I(n); j = J(n);
78         if canlink0(mtx,i,j,x2,y2)
79             bool = 1;
80             return
81         end
82     end
83
84     bool = 0;
85
86 end

```

Listing 2.3: canlink.m(canlink1)

(d) **判断可连性 canlink** : 选取两目标点之一作为起点，画十字；对十字上的点进行遍历，检查是否存在可与另一目标用不超过一个直角的连线连接的点。

```

1 function bool = canlink(mtx,x1,y1,x2,y2)
2 % return 1 if these two blocks can be linked!
3     if mtx(x1,y1) ~= mtx(x2,y2) || ~mtx(x1,y1) || ~mtx(x2,y2)
4         bool = 0;
5         return
6     end
7
8     if canlink1(mtx,x1,y1,x2,y2)
9         bool = 1;
10        return
11    end
12
13    % grow the cross of origin
14    [I,J] = adjcross(mtx,x1,y1);
15
16    for n = 1:length(I)
17        i = I(n); j = J(n);

```

```

18         if canlink1(mtx,i,j,x2,y2)
19             bool = 1;
20             return
21         end
22     end
23
24     bool = 0;
25
26 end

```

Listing 2.4: canlink.m(main)

再次运行 linkgame, detect.m功能正常。

```

1  function bool = detect(mtx, x1, y1, x2, y2)
2
3      [m,n] = size(mtx);
4
5      % add surrounding zeros
6      mtx = [0,zeros(1,n),0;
7             zeros(m,1),mtx,zeros(m,1);
8             0,zeros(1,n),0];
9
10     origin = mtx(x1+1,y1+1);
11     target = mtx(x2+1,y2+1);
12
13     if origin == target && canlink(mtx,x1+1,y1+1,x2+1,y2+1)
14         bool = 1;
15     else
16         bool = 0;
17     end
18
19 end

```

Listing 2.5: detect.m(main)

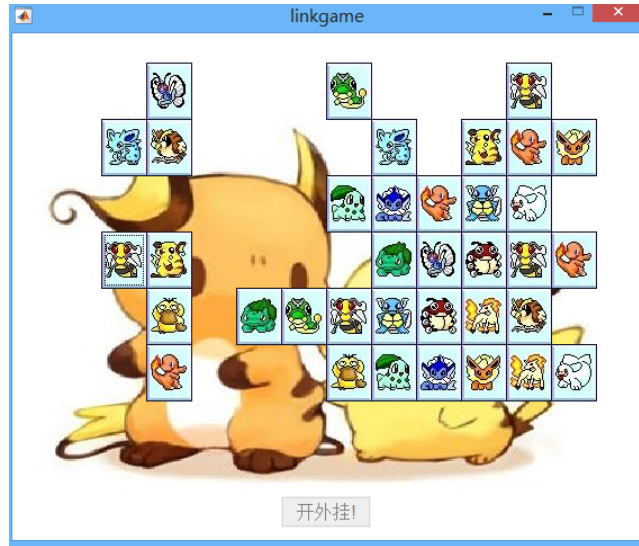


Figure 2.2: 重写 detect.m 后运行 linkgame

3. “外挂”模式逐一自动消除所有的块的功能是由 link 目录的 omg.p 实现的。删掉 omg.p 重新实现 omg.m。

根据游戏经验，算法通过以下几个步骤实现：

(a) 消去相邻的相同块 matchadj：利用自带函数 diff 实现，注意及时更新原矩阵以及保证多个相同块连续相邻时仍正确工作。

```

42 function [steps,mtx] = matchadj(mtx)
43 % match adjacent removable blocks
44
45 steps = [];
46
47 % match adjacent blocks
48 % using diff() is effective
49 % row difference
50 rdif = diff(mtx,1,1);
51 [I,J] = find(rdif==0);
52 for n = 1:length(I)
53     % empty blocks
54     if mtx(I(n),J(n)) ~= 0 && mtx(I(n)+1,J(n)) ~= 0
55         % match!
56         steps = [steps,I(n),J(n),I(n)+1,J(n)];
57         % update mtx
58         mtx(I(n),J(n)) = 0;

```



```

59         mtx(I(n)+1,J(n)) = 0;
60     end
61 end
62
63 % column difference
64 cdiff = diff(mtx,1,2);
65 [I,J] = find(cdiff==0);
66 for n = 1:length(I)
67     % ?empty blocks
68     if mtx(I(n),J(n)) ~= 0 && mtx(I(n),J(n)+1) ~= 0
69         % match!
70         steps = [steps,I(n),J(n),I(n),J(n)+1];
71         % update mtx
72         mtx(I(n),J(n)) = 0;
73         mtx(I(n),J(n)+1) = 0;
74     end
75 end
76 end

```

Listing 2.6: omg.m(matchadj)



Figure 2.3: 测试 matchadj() 功能

matchadj 函数工作正常。

- (b) 消去同一条边界上的相同块 matchborder : 先定义上边界,如图2.4中的蓝色区域。



Figure 2.4: 定义上边界

显然，上边界即每一列第一个非零元素；类似地可以定义其他边界。

容易证明，在同一边界上的相同块必然可消去。

与 `matchadj` 不同，由于消去过程会使边界发生变化，故必须不断循环直至边界保持不变。则实现 `matchborder` 函数如下：

```

79 function [steps,mtx] = matchborder(mtx)
80 % match blocks on the same border
81
82 [m,n] = size(mtx);
83 steps = [];
84
85 isstable = 0; % whether mtx is stable, i.e. no more removable pairs on borders
86 while isstable ~= 4 % stable on four borders
87     for k = 1:4 % four borders
88         if k == 1
89             % upper border
90             ub = sum(cumsum(mtx)==0) + 1;
91             ub(ub>m) = nan; % a column of zeros
92             index = sub2ind(size(mtx),ub,1:n);
93         elseif k == 2
94             % bottom border
95             bb = m - sum(cumsum(mtx,'reverse')==0);
96             bb(bb<1) = nan;
97             index = sub2ind(size(mtx),bb,1:n);
98         elseif k == 3

```

```

99         % left border
100         lb = sum(cumsum(mtx,2)==0,2) + 1;
101         lb(lb>n) = nan;
102         index = sub2ind(size(mtx),1:m,lb. ');
103     else
104         % right border
105         rb = n - sum(cumsum(mtx,2,'reverse')==0,2);
106         rb(rb<1) = nan;
107         index = sub2ind(size(mtx),1:m,rb. ');
108     end
109     index = index(~isnan(index)); % delete nan
110     blocks = mtx(index);
111
112     b = unique(blocks);
113     if length(b) < length(blocks) % same blocks
114         isstable = 0;
115         for be = b
116             i = find(blocks==be);
117             if length(i) >= 2
118                 [I,J] = ind2sub(size(mtx),index(i(1:2)));
119                 % add steps
120                 steps = [steps,I(1),J(1),I(2),J(2)];
121                 % update mtx
122                 mtx(index(i(1:2))) = 0;
123                 break
124             end
125         end
126     else
127         isstable = isstable + 1;
128         if isstable == 4
129             break
130         end
131     end
132 end
133 end
134
135 end

```

Listing 2.7: omg.m(matchborder)

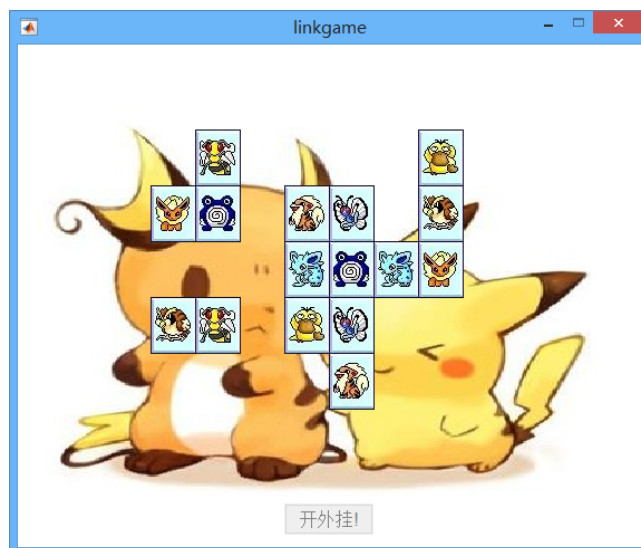


Figure 2.5: 依次调用 matchadj 和 matchborder 后

经过多次测试，发现 matchadj 与 matchborder 相结合的算法已经能消去游戏区域的**绝大多数块**，甚至有时可**全部消去**。

matchadj 的核心是 MATLAB 自带的 diff 函数，效率较高；

而 matchborder 通过 cumsum 以及 sum 函数巧妙获得各边界索引，再利用 unique 以及 find 等方法在边界内寻找相同的块，可以说十分高效。

既然高效的前两步已可以消去大多数块，那么对于剩余的块，不妨采取较暴力的算法解决。

- (c) 对于剩余的块按种类遍历尝试连接 matchrest：这里假设生成的连连看游戏是可以以任意消除顺序完全消除的（实践观测结果如此）。

```

138 function [steps,mtx] = matchrest(mtx)
139 % match rest blocks
140
141 steps = [];
142
143 while any(any(mtx)) % game NOT over
144     kinds = unique(mtx); % get kinds of blocks
145     kinds = kinds(2:end); % remove 0
146
147     for k = 1:length(kinds)
148         kind = kinds(k);
149         [I,J] = find(mtx==kind);

```

```

150         for m = 1:length(I)
151             for n = m+1:length(I)
152                 if canlink(mtx,I(m),J(m),I(n),J(n))
153                     steps = [steps,I(m),J(m),I(n),J(n)];
154                     % update mtx
155                     mtx(I(m),J(m)) = 0;
156                     mtx(I(n),J(n)) = 0;
157                     break
158                 end
159             end
160         end
161     end
162 end
163
164 end

```

Listing 2.8: omg.m(matchrest)

经过多次测试，均能顺利完成功能。

```

1 function steps = omg(mtx)
2
3     [m,n] = size(mtx);
4
5     % add surrounding zeros
6     mtx = [0,zeros(1,n),0;
7           zeros(m,1),mtx,zeros(m,1);
8           0,zeros(1,n),0];
9
10    [steps1,mtx] = matchadj(mtx);
11    [steps2,mtx] = matchborder(mtx);
12    [steps3,mtx] = matchrest(mtx);
13    steps = [steps1,steps2,steps3];
14
15    % make steps meet interface
16    steps = [length(steps)/4,steps-1];
17
18 end

```

Listing 2.9: omg.m(main)

3 攻克别人的连连看

1. 在 MATLAB 环境下，将路径设置到 `process` 文件夹下。对游戏区域的屏幕截图 (灰度图像) `graygroundtruth` 进行分割，提取所有图像分块。

先进行预处理：

```
1 %% load image and preprocess
2 original = imread('graygroundtruth.jpg');
3 image = double(original);
4 image = image - 128;
```

绘制水平竖直扫描线灰度均值如图3.1。

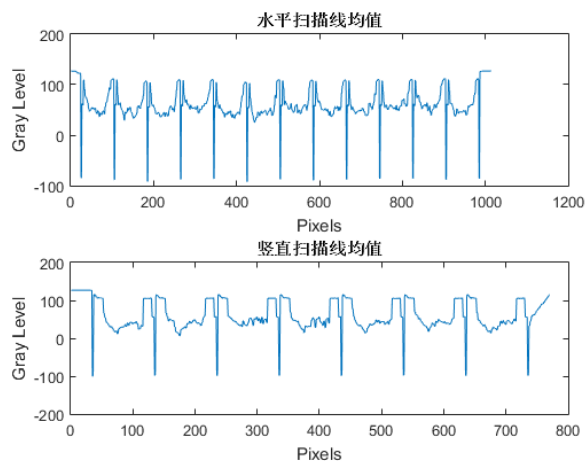


Figure 3.1: 扫描线灰度均值

周期如此明显，可以通过一些简单的运算直接获得块大小 (w_b, h_b)、游戏区域位置信息 (x_s, y_s) 以及块数量 (N_c, N_r) (而不需要进行傅里叶变换)。

```
1 %% get block size and position of game region
2 hor = mean(image,1);
3 ver = mean(image,2);
4
5 npeak = min(hor); % negative peak
6 I = find(hor < 0.9*npeak);
7 I = sort(I);
8 Xs = I(1);
9 I = diff(I); % width between peaks
```

```

10 I = I(I>30); % filter too small blocks
11 Wb = mean(I)+1;
12 Nc = length(I); % number of block columns
13
14 npeak = min(ver); % negative peak
15 I = find(ver<0.9*npeak);
16 I = sort(I);
17 Ys = I(1);
18 I = diff(I); % width between peaks
19 I = I(I>30); % filter too small blocks
20 Hb = mean(I)+1;
21 Nr = length(I); % number of block rows

```

Listing 3.1: divide.m 获取块大小以及位置信息

然后按得到的尺寸 (Xs=25, Ys=35, Wb=80, Hb=100, Nc=12, Nr=7) 获取各图像块, 效果良好 :

```

20 blocks = {};
21 figure(2);
22 for i = 1:Nr
23     for j = 1:Nc
24         block = original(round(Ys+(i-1)*Hb+1):round(Ys+i*Hb),...
25             round(Xs+(j-1)*Wb+1):round(Xs+j*Wb));
26         blocks{end+1} = block;
27         subplot(Nr,Nc,(i-1)*Nc+j);
28         imshow(block);
29     end
30 end

```

Listing 3.2: divide.m 获取所有块



Figure 3.2: 图像分割效果

2. 对摄像头采集的图像 (灰度图像) `graycapture`, 参考第 1 题要求进行处理。

摄像头采集的图像噪音明显增多, 而且图像有小幅度的旋转。上一题使用的方法在这里完全不能获得正确结果。

另外, 画面对比度较弱, 使得横纵方向的周期性也不明显了。

考虑将其转化为二值图像 (黑白图像) 以增强周期性。

预处理 :

```
1 %% load image and preprocess
2 original = imread('graycapture.jpg');
3 image = im2bw(original,0.8); % convert to bw image
4 image = double(image);
```

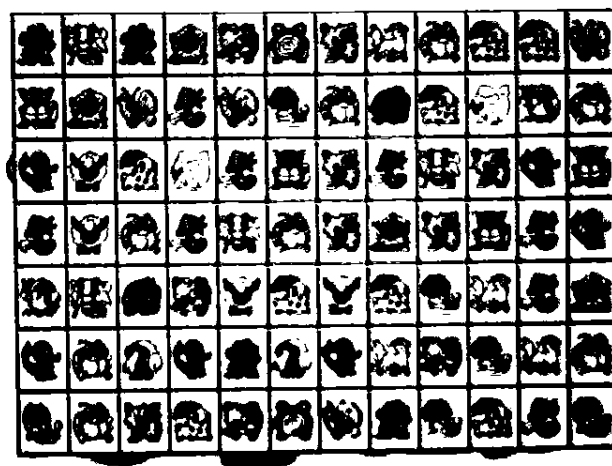



Figure 3.3: graycapture 预处理成黑白图像

再观察扫描线均值：

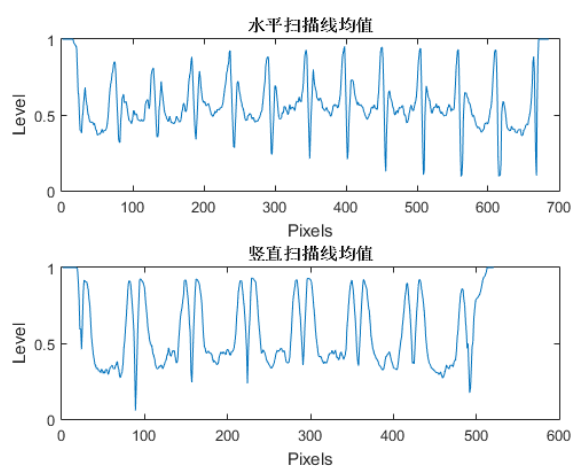


Figure 3.4: 扫描线黑白均值

周期性已经比较明显，块分界线（图3.3中的黑色直线）在扫描线黑白均值3.4中体现为尖锐的低谷。

考虑用 `findpeaks` 函数获取这些低谷的位置，测试出性能较好的参数如下：

Table 3.1: findpeaks 函数参数设置

选项	值	作用
MinPeakProminence	0.3	筛选出较突出 (尖锐) 的峰值
MaxPeakWidth	10	去除过宽的峰值
MinPeakDistance	30	去除距离过近的峰值

找到的峰值如下：

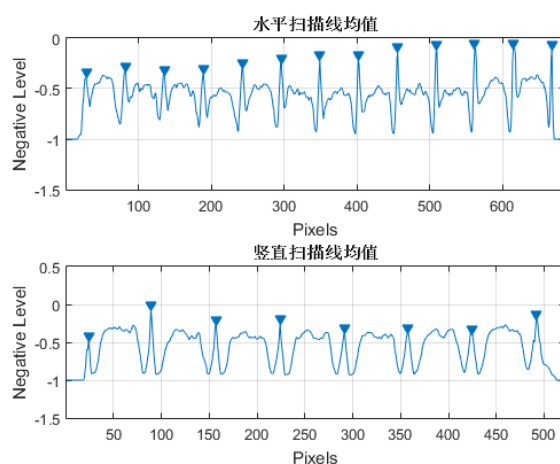


Figure 3.5: findpeaks 检测峰值 (纵坐标为原数据的相反数)

可见效果相当好，免去繁杂的傅里叶变换 (若不能得当地处理噪声，傅里叶变换得到的结果不是特别准确)。

```

1  %% get block size and position of game region
2  hor = mean(image,1);
3  ver = mean(image,2);
4
5  [~,I] = findpeaks(-hor,'MinPeakProminence',0.3,'MaxPeakWidth',10,'MinPeakDistance',30);
6  [~,J] = findpeaks(-ver,'MinPeakProminence',0.3,'MaxPeakWidth',10,'MinPeakDistance',30);
7  Xs = I(1); Ys = J(1);
8  Wb = mean(diff(I)); Nc = length(I) - 1;
9  Hb = mean(diff(J)); Nr = length(J) - 1;

```

得到结果 (Xs=28, Ys=24, Wb=53.33, Hb=66.86, Nc=12, Nr=7)，仍然运行代码3.2绘制获取到的所有块。



Figure 3.6: 利用二值图像法得到的图像分割效果

分割效果很好。若使用**二值图像法**重新对第一题的 `graygroundtruth` 进行处理，得到的尺寸参数为 ($X_s=25$, $Y_s=35$, $W_b=80$, $H_b=100$, $N_c=12$, $N_r=7$)，与使用第一题方法得到的结果完全相同。可见**二值图像法**对于干净图像以及摄像头采集图像均有良好的性能。

二值图像法利用黑白图像的强对比度增强了图像的周期性，便于后续提取尺寸信息；若不转化为黑白图像，直接傅里叶变换可以看到基波频率附近有许多能量也很高的噪声频率分量，对识别基波频率的准确度有巨大影响。**二值图像法**虽然丢失了图像的细节，但增强了对分块影响最大的周期性，因此**二值图像法**用于分块不失为一种优越的方法。

3. 在第 2 题基础上，计算所有图像分块的两两相似性，选出最相似的十对图像块。

(a) **高通滤波**：利用 `imfilter` 函数对图像进行高通滤波提取纹理特征。

```

1  % ...
2
3  hp = [-0.3,-0.3,-0.3;
4        -0.3, 0.5,-0.3;
5        -0.3,-0.3,-0.3];    % construct high pass filter
6
7  N = length(blocks);
8  for i = 1:N
9      im1 = imfilter(double(blocks{i})-128, hp);
10     im1 = (im1-min(min(im1)))/(max(max(im1))-min(min(im1)))*255;
11
12  % ...

```

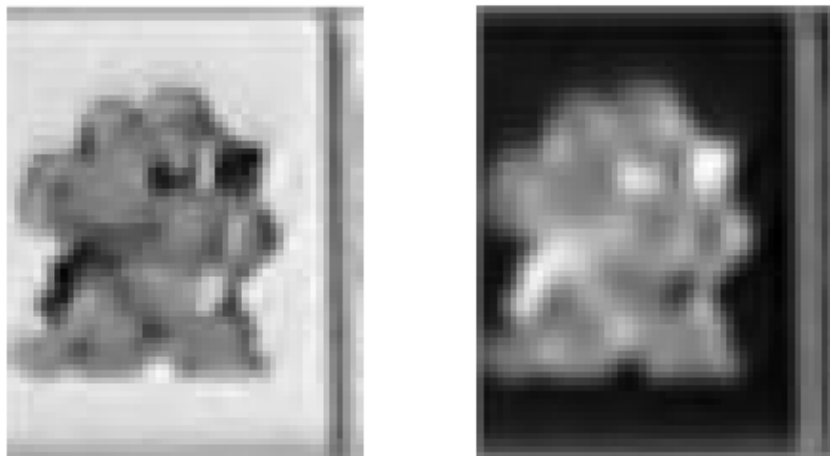


Figure 3.7: 高通滤波提取纹理特征

(b) 计算两图像相似度：匹配滤波。

```

1 function S = similarity(im1,im2)
2 % return similarity of two intensity images: im1 and im2
3 % notice: 0 <= S <= 1
4
5     im1 = double(im1);
6     im2 = double(im2);
7     S = max(max(filter2(im1,im2)) / ...
8         sqrt(sum(sum(im1.^2))*sum(sum(im2.^2))));
9
10 end

```

Listing 3.3: similarity.m

(c) 两两计算相似度并排序：二重循环暴力实现。

```

1 function match_map = similarsort(blocks)
2 % <1-by-n cell> blocks
3 % <m-by-3 matrix> match_map: m == n*(n-1)/2,
4 %     the first 2 elements are indexes of blocks,
5 %     the third element is similarity
6 %     rows of match_map are sorted by similarity
7
8     match_map = [];
9

```

```

10 hp = [-0.3,-0.3,-0.3;
11        -0.3, 0.5,-0.3;
12        -0.3,-0.3,-0.3]; % construct high pass filter
13
14 N = length(blocks);
15 for i = 1:N
16     im1 = imfilter(double(blocks{i})-128,hp);
17     im1 = (im1-min(min(im1)))/(max(max(im1))-min(min(im1)))*255-128;
18     for j = i+1:N
19         im2 = imfilter(double(blocks{j})-128,hp);
20         im2 = (im2-min(min(im2)))/(max(max(im2))-min(min(im2)))*255-128;
21         match_map = [match_map; i,j,similarity(im1,im2)];
22     end
23 end
24
25 match_map = sortrows(match_map,-3);
26
27 end

```

Listing 3.4: similarsort.m

尽量使各图像块的灰度值均匀地分布在 0 的两侧 (-128,127)，这样做相似度运算时，正负相消使得结果更准确了。

分别用 `graygroundtruth` 和 `graycapture` 测试，按相似度排序后，相似度走势曲线如图3.8。(两个 `match_map` 分别存在 `graygroundtruth_match_map.mat`和`graycapture_match_map.mat`中)

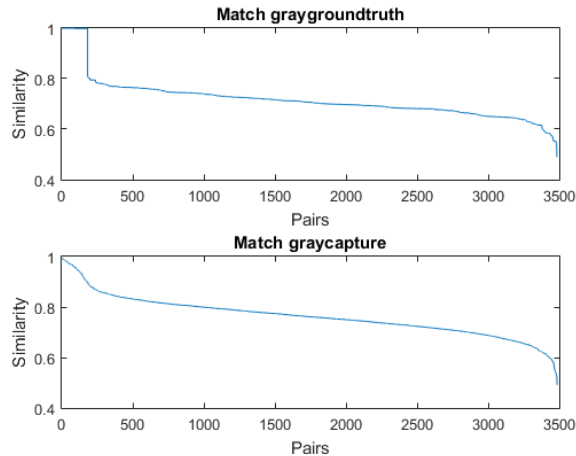


Figure 3.8: 相似度走势曲线

- 对于屏幕截图 `graygroundtruth`, 相同两块的相似度几乎均为 1, 而不同两块的相似度明显较低 (低于 0.8);
- 对于摄像头采集图像 `graycapture`, 相似度曲线几乎是连续的, 但在 `Similarity=0.88` 附近也出现了明显的拐点。

找出 `graycapture` 中相似度最高的十对图像块 :

```
1 figure;
2
3 for i = 1:10
4     subplot('position',[0.1+(i-1)*0.08,0.55,0.08,0.1]);
5     imshow(blocks{match_map2(i,1)}); title(num2str(match_map2(i,3)));
6     subplot('position',[0.1+(i-1)*0.08,0.35,0.08,0.1]);
7     imshow(blocks{match_map2(i,2)});
8 end
```

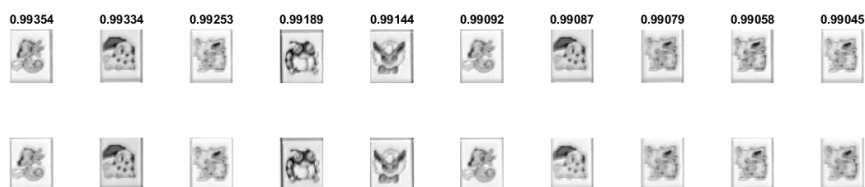


Figure 3.9: 相似度最高的十对图像块

相似度最高的十对图像块确实是相同的图像块。

4. 在第 3 题的基础上, 找出相似度最大却不是同一种精灵的十对图像块。

首先需手动标定各位置的精灵类别 :

```
1 kind = [1, 2, 1, 3, 4, 5, 6, 7, 8, 9, 9, 10;
2         11, 3, 10, 12, 10, 13, 8, 14, 9, 15, 16, 8 ;
3         17, 18, 9, 15, 12, 11, 6, 12, 2, 6, 17, 11;
4         12, 18, 8, 12, 2, 8, 6, 3, 6, 11, 12, 17;
5         16, 2, 14, 4, 18, 9, 18, 9, 13, 7, 12, 3 ;
6         17, 8, 19, 17, 1, 19, 17, 7, 4, 13, 7, 8 ;
7         13, 8, 6, 9, 4, 5, 10, 1, 13, 9, 12, 13;];
```

Listing 3.5: 手动标定精灵种类

找出十对不同但相似度最高的图像块 :

Table 3.2: 精灵种类对应表 (人脑识别)

序号	精灵
1	小锯鳄
2	大针蜂
3	水精灵
4	喵喵
5	蚊香蛙
6	尼多兰
7	小火马
8	芭瓢虫
9	菊草叶
10	巴大蝴
11	杰尼龟
12	墨海马
13	绿毛虫
14	妙蛙种子
15	小海狮
16	波波
17	小火龙
18	火精灵
19	卡蒂狗

```

1 figure;
2 count = 0;
3 kindt = kind.';
4
5 for i = 1:size(match_map2,1)
6     if kindt(match_map2(i,1)) ~= kindt(match_map2(i,2))
7         count = count + 1;
8         subplot('position',[0.1+(count-1)*0.08,0.55,0.08,0.1]);
9         imshow(blocks{match_map2(i,1)}); title(num2str(match_map2(i,3)));
10        subplot('position',[0.1+(count-1)*0.08,0.35,0.08,0.1]);
11        imshow(blocks{match_map2(i,2)});
12        xlabel(['index:',num2str(i)]);
13        if count == 10
14            break
15        end
16    end
17 end

```

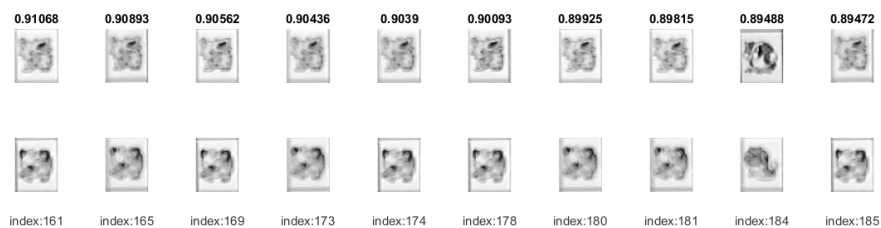


Figure 3.10: 不同块最相似前十名

图3.10在每对图像块下方标注了它们在排序后 `match_map2` 中的位置，这说明并非从某一位置开始起，后面的图像块均是不同块，至少在相似度 (0.89, 0.91) 这一区间内，正确和错误的判断掺杂在了一起，这表明摄像头采集图像的噪音使得算法可能发生误判。而屏幕截图很少出现这样的问题。

图3.10的结果和主观感受部分符合，惊人地发现前十名误判中有 90% 是尼多兰和喵喵，原因可能是轮廓较像且摄像头拍摄模糊 (高通滤波使得细节丢失了)。

既然高通滤波丢失了一些细节信息，那么不做高通滤波，直接做相似度计算，结果如何呢？对代码稍作修改，重复上述过程得到：

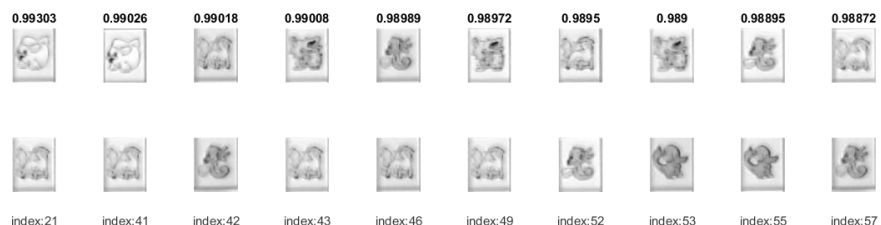


Figure 3.11: 不做高通滤波直接利用相似度匹配的误判前十名

不做高通滤波使得重要的纹理特征被掩盖了，细节喧宾夺主，在相似度高达 0.99 时就发生了误判。这说明高通滤波是至关重要的一步。

这也从另一方面说明了：使用不同的高通滤波器也对判断结果有影响。

5. 在第 3 题基础上，将游戏区域映射为索引值的数组，并列出索引值和图像分块的对照关系表。

在上一题中, 已经手动做过这一步 (表3.2以及代码3.5), 现用程序实现, 比较结果。

分类函数 `classify` :

```
1 function category = classify(match_map,threshold,N)
2 % <m-by-3 matrix> match_map: match_map(i,3) = similarity
3 % <int> N: number of elements to be classified
4 % classify by similarity
5 % they're in the same class if similarity > threshold
6
7 % <n-by-1 cell> category: n CLASSES
8
9 category{1} = match_map(1,1:2);
10
11 isclassified = zeros(1,N); % mark CLASS of each element
12 isclassified(match_map(1,1:2)) = 1; % CLASS 1
13
14 for i = 1:size(match_map,1)
15     row = match_map(i,:);
16     if row(3) < threshold
17         indexes = find(isclassified==0);
18         for index = indexes
19             [J,I] = find(match_map.'==index,10); % find first 10 pairs containing INDEX
20             inds = sub2ind(size(match_map),I,3-J);
21             class = mode(isclassified(match_map(inds)));
22             category{class} = union(category{class},match_map(I(1),J(1)));
23             isclassified(match_map(I(1),J(1))) = class;
24         end
25         % finished
26         break
27     end
28
29     status = isclassified(row(1:2)); % classify status
30     if sum(status) == 0
31         % all not classified
32         % new class
33         category{end+1} = row(1:2);
34         isclassified(row(1:2)) = length(category);
35     elseif all(status~=0)
36         % all classified
37         class1 = isclassified(row(1));
38         class2 = isclassified(row(2));
39         if class1 ~= class2
40             % merge class2 into class1
41             e = category{class2};
```

```

42         category{class1} = [category{class1},e];
43         isclassified(e) = class1;
44         category{class2} = [];
45     end
46 else
47     % one classified, the other NOT
48     if status(1) == 0
49         class = isclassified(row(2));
50         category{class} = [category{class},row(1)];
51         isclassified(row(1)) = class;
52     else
53         class = isclassified(row(1));
54         category{class} = [category{class},row(2)];
55         isclassified(row(2)) = class;
56     end
57 end
58 end
59
60 % delete empty cells
61 result = {};
62 for i = 1:length(category)
63     if ~isempty(category{i})
64         result{end+1} = category{i};
65     end
66 end
67
68 category = result;
69
70 end

```

Listing 3.6: classify.m

算法核心为以下几点：

- 维护 `isclassified` 向量，长度为 N (要分类元素总数)，每个位置存储对应元素当前的类别，尚未分类用 0 表示；
- 设定阈值 `threshold`，只对相似度高于阈值的 `Pair` 进行分类操作；
- 具体分类时，若当前 `Pair` 的两个元素均未分类，则开辟新的类别；若两元素之一已分类，则将另一元素也加入该分类；若两元素已分至同一类别，则跳过；若两元素已分至不同类别，则将两类别合并；
- 高于阈值的 `Pair` 全部处理完仍有元素未分类，则对未分类的元素遍历，寻找其相似度最高的分类添加。

根据图3.8可确定合适的阈值：

```

1 >> category1 = classify(match_map1,0.9,84);
2 >> category2 = classify(match_map2,0.92,84);

```

分别对 graygroundtruth 和 graycapture 分类的结果均为 19 个类别。
以 graycapture 为例可视化：

```

1 figure;
2
3 for i = 1:length(category2)
4     subplot(5,5,i);
5     C = category2{i};
6     imshow(blocks{C(1)});
7     title(['#',num2str(i),' Counts: ',num2str(length(C))]);
8 end

```

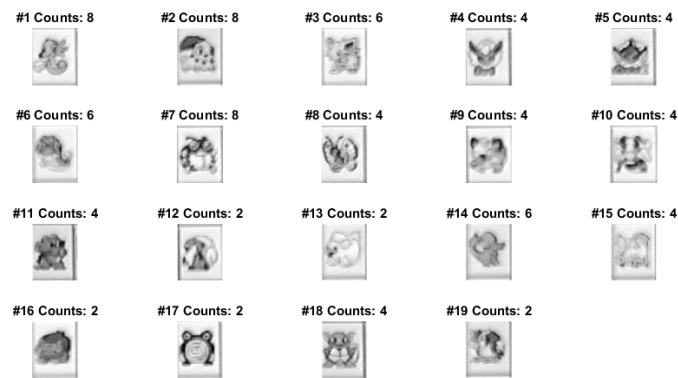


Figure 3.12: 对 graycapture 自动分类结果

与手动分类结果完全一致，正确率 100%。

至此，可以得到游戏区域的 mtx 矩阵：

```

1 mtx = zeros(Nc,Nr);
2
3 for i = 1:length(category2)
4     mtx(category2{i}) = i; % rowwise index
5 end
6
7 mtx = mtx.';

```

6. 在上述工作基础上, 设计实现一个模拟的自动连连看。对摄像头采集的图像 (灰度图像) `graycapture` 进行分块并找出最相似的一对可消除分块后, 将这图片上两个块的位置设为黑色或其他特定颜色 (即模拟消除操作), 并将图片展示在 `figure` 上。然后继续找出下一对可消除的分块并模拟消除, 直至消除所有的分块或找不到可消除的分块对。

首先对图像进行处理, 获得游戏区域的 `mtx` 矩阵:

```
1  %% run divide.m
2  divide;
3
4
5  %% calculate similarity
6  match_map = similarsort(blocks);
7
8
9  %% classify
10 category = classify(match_map,0.92,Nc*Nr);
11
12
13 %% get mtx
14 mtx = zeros(Nc,Nr);
15
16 for i = 1:length(category)
17     mtx(category{i}) = i; % rowwise index
18 end
19
20 mtx = mtx.';
21
22 % ...
```

Listing 3.7: `mocklink.m`(1-22, get `mtx`)

和之前的 `canlink.m` 不同, 这里进行模拟连接时不仅要考虑是否可以连接, 我们还希望函数能够返回相应的连接路径, 进而绘制出连接路径, 这样模拟连接的过程更真实。

对 `canlink` 函数稍作修改得到 `findpath` 函数:

```
1 function [canlink,nodesX,nodesY] = findpath(mtx,x1,y1,x2,y2)
2 % Update function CANLINK to FINDPATH, returning path nodes as well
3 % return [1,nodesX,nodesY] if these two blocks can be linked
4
5     nodesX = [];
```

```

6     nodesY = [];
7
8     if mtx(x1,y1) ~= mtx(x2,y2) || ~mtx(x1,y1) || ~mtx(x2,y2)
9         canlink = 0;
10        return
11    end
12
13    [canlink,nodesX,nodesY] = findpath1(mtx,x1,y1,x2,y2);
14    if canlink
15        return
16    end
17
18    % grow the cross of origin
19    [I,J] = adjcross(mtx,x1,y1);
20
21    for n = 1:length(I)
22        i = I(n); j = J(n);
23        [canlink,nodesX,nodesY] = findpath1(mtx,i,j,x2,y2);
24        if canlink
25            nodesX = [x1,nodesX];
26            nodesY = [y1,nodesY];
27            return
28        end
29    end
30
31    canlink = 0;
32
33 end
34
35
36 function [I,J] = adjcross(mtx,x,y)
37 % return index vector of can-reach blocks of mtx(x,y) on both directions
38 % notice: x >= 2 && y >= 2 && all(mtx >= 0)
39
40 % get vectors of four directions
41 left = mtx(x,1:y-1);
42 right = mtx(x,y+1:end);
43 up = mtx(1:x-1,y);
44 down = mtx(x+1:end,y);
45
46 % get zero run length adjacent to mtx(x,y)
47 lz = find(cumsum(left,'reverse')==0);
48 rz = find(cumsum(right,'forward')==0);
49 uz = find(cumsum(up,'reverse')==0);
50 dz = find(cumsum(down,'forward')==0);
51

```

```

52     I = [x*ones(1,length([lz,rz])),uz.',x+dz.'];
53     J = [lz,y+rz,y*ones(1,length([uz,dz]))];
54
55     end
56
57
58     function [bool,nodesX,nodesY] = findpath0(mtx,x1,y1,x2,y2)
59         % return [1,nodesX,nodesY] if it's a direct link (no turns)
60         % return [0,~,~] otherwise
61
62         nodesX = [];
63         nodesY = [];
64
65         if x1 == x2 && ~any(mtx(x1,min(y1,y2)+1:max(y1,y2)-1))
66             bool = 1;
67             nodesX = [x1,x2];
68             nodesY = [y1,y2];
69         elseif y1 == y2 && ~any(mtx(min(x1,x2)+1:max(x1,x2)-1,y1))
70             bool = 1;
71             nodesX = [x1,x2];
72             nodesY = [y1,y2];
73         else
74             bool = 0;
75         end
76
77     end
78
79
80     function [bool,nodesX,nodesY] = findpath1(mtx,x1,y1,x2,y2)
81         % return [1,nodesX,nodesY] if the turns of link path <= 1
82         % return [0,~,~] otherwise
83
84         [bool,nodesX,nodesY] = findpath0(mtx,x1,y1,x2,y2);
85         if bool
86             return
87         end
88
89         % grow the cross of origin
90         [I,J] = adjcross(mtx,x1,y1);
91
92         for n = 1:length(I)
93             i = I(n); j = J(n);
94             [bool,nodesX,nodesY] = findpath0(mtx,i,j,x2,y2);
95             if bool
96                 nodesX = [x1,nodesX];
97                 nodesY = [y1,nodesY];

```

```

98         return
99     end
100 end
101
102 bool = 0;
103 nodesX = [];
104 nodesY = [];
105
106 end

```

Listing 3.8: findpath.m

相应地，将 omg.m 修改成为 `solverlinkgame.m`，增加了 `connections` 返回值，存储每次消除时连线的结点：

```

1  function [steps,connections] = solverlinkgame(mtx)
2
3      [m,n] = size(mtx);
4
5      % add surrounding zeros
6      mtx = [0,zeros(1,n),0;
7             zeros(m,1),mtx,zeros(m,1);
8             0,zeros(1,n),0];
9
10     [steps1,connections1,mtx] = matchadj(mtx);
11     [steps2,connections2,mtx] = matchborder(mtx);
12     [steps3,connections3,mtx] = matchrest(mtx);
13     steps = [steps1,steps2,steps3];
14     connections = [connections1;connections2;connections3];
15
16     % make steps meet interface
17     steps = [length(steps)/4,steps-1];
18
19 end
20
21
22 function [steps,connections,mtx] = matchadj(mtx)
23 % match adjacent removable blocks
24
25     steps = [];
26     connections = {};
27
28     % match adjacent blocks
29     % using diff() is effective
30     % row difference
31     rdifff = diff(mtx,1,1);

```

```

32     [I,J] = find(rdiff==0);
33     for n = 1:length(I)
34         % ?empty blocks
35         if mtx(I(n),J(n)) ~= 0 && mtx(I(n)+1,J(n)) ~= 0
36             % match!
37             steps = [steps,I(n),J(n),I(n)+1,J(n)];
38             connections = [connections;{[I(n),I(n)+1]-1,[J(n),J(n)]-1}];
39             % update mtx
40             mtx(I(n),J(n)) = 0;
41             mtx(I(n)+1,J(n)) = 0;
42         end
43     end
44
45     % column difference
46     cdiff = diff(mtx,1,2);
47     [I,J] = find(cdiff==0);
48     for n = 1:length(I)
49         % ?empty blocks
50         if mtx(I(n),J(n)) ~= 0 && mtx(I(n),J(n)+1) ~= 0
51             % match!
52             steps = [steps,I(n),J(n),I(n),J(n)+1];
53             connections = [connections;{[I(n),I(n)]-1,[J(n),J(n)+1]-1}];
54             % update mtx
55             mtx(I(n),J(n)) = 0;
56             mtx(I(n),J(n)+1) = 0;
57         end
58     end
59 end
60
61
62 function [steps,connections,mtx] = matchborder(mtx)
63 % match blocks on the same border
64
65 [m,n] = size(mtx);
66 steps = [];
67 connections = {};
68
69 isstable = 0; % whether mtx is stable, i.e. no more removable pairs on borders
70 while isstable ~= 4 % stable on four borders
71     for k = 1:4 % four borders
72         if k == 1
73             % upper border
74             ub = sum(cumsum(mtx)==0) + 1;
75             ub(ub>m) = nan; % a column of zeros
76             index = sub2ind(size(mtx),ub,1:n);
77         elseif k == 2

```



```

78         % bottom border
79         bb = m - sum(cumsum(mtx,'reverse')==0);
80         bb(bb<1) = nan;
81         index = sub2ind(size(mtx),bb,1:n);
82     elseif k == 3
83         % left border
84         lb = sum(cumsum(mtx,2)==0,2) + 1;
85         lb(lb>n) = nan;
86         index = sub2ind(size(mtx),1:m,lb. ');
87     else
88         % right border
89         rb = n - sum(cumsum(mtx,2,'reverse')==0,2);
90         rb(rb<1) = nan;
91         index = sub2ind(size(mtx),1:m,rb. ');
92     end
93     index = index(~isnan(index)); % delete nan
94     blocks = mtx(index);
95
96     b = unique(blocks);
97     if length(b) < length(blocks) % same blocks
98         isstable = 0;
99         for be = b
100             i = find(blocks==be);
101             if length(i) >= 2
102                 [I,J] = ind2sub(size(mtx),index(i(1:2)));
103                 % add steps
104                 steps = [steps,I(1),J(1),I(2),J(2)];
105                 [~,nodesX,nodesY] = findpath(mtx,I(1),J(1),I(2),J(2));
106                 connections = [connections;{nodesX-1,nodesY-1}];
107                 % update mtx
108                 mtx(index(i(1:2))) = 0;
109                 break
110             end
111         end
112     else
113         isstable = isstable + 1;
114         if isstable == 4
115             break
116         end
117     end
118 end
119 end
120
121 end
122
123

```

```

124 function [steps,connections,mtx] = matchrest(mtx)
125 % match rest blocks
126
127     steps = [];
128     connections = {};
129
130     while any(any(mtx)) % game NOT over
131         kinds = unique(mtx); % get kinds of blocks
132         kinds = kinds(2:end); % remove 0
133
134         for k = 1:length(kinds)
135             kind = kinds(k);
136             [I,J] = find(mtx==kind);
137             for m = 1:length(I)
138                 for n = m+1:length(I)
139                     [canlink,nodesX,nodesY] = findpath(mtx,I(m),J(m),I(n),J(n));
140                     if canlink
141                         steps = [steps,I(m),J(m),I(n),J(n)];
142                         connections = [connections;{nodesX-1,nodesY-1}];
143                         % update mtx
144                         mtx(I(m),J(m)) = 0;
145                         mtx(I(n),J(n)) = 0;
146                         break
147                     end
148                 end
149             end
150         end
151     end
152
153 end

```

Listing 3.9: solvelinkgame.m

最后是比较繁琐的可视化过程：

```

22 % ...
23 %% solve mtx
24 [steps,connections] = solvelinkgame(mtx);
25
26
27 %% visualization
28 % show original picture
29 figure(1);
30 imshow(original);
31 hold on;
32 pause(5);

```

```

33
34 % visualize steps
35 image = original;
36 [height,width] = size(image);
37 for n = 1:size(connections,1)
38     pre_image = image; % save previous image
39     connection = connections(n,:);
40     nodesX = connection{1};
41     nodesY = connection{2};
42     for m = 1:length(nodesX)-1
43         x1 = round(Ys + (min(nodesX(m),nodesX(m+1))-1)*Hb + Hb/2);
44         x2 = round(Ys + (max(nodesX(m),nodesX(m+1))-1)*Hb + Hb/2);
45         y1 = round(Xs + (min(nodesY(m),nodesY(m+1))-1)*Wb + Wb/2);
46         y2 = round(Xs + (max(nodesY(m),nodesY(m+1))-1)*Wb + Wb/2);
47         x1 = min(max(x1,round(Ys/2)),round((height+Ys+Nr*Hb)/2));
48         y1 = min(max(y1,round(Xs/2)),round((width+Xs+Nc*Wb)/2));
49         x2 = min(max(x2,round(Ys/2)),round((height+Ys+Nr*Hb)/2));
50         y2 = min(max(y2,round(Xs/2)),round((width+Xs+Nc*Wb)/2));
51
52         image(x1:x2+2,y1:y2+2) = 0; % color=black, width=3
53     end
54     % show link
55     imshow(image);
56     pause(1);
57
58     image = pre_image;
59     x1 = round(Ys + (steps(1+(n-1)*4+1)-1)*Hb);
60     y1 = round(Xs + (steps(1+(n-1)*4+2)-1)*Wb);
61     x2 = round(x1 + Hb);
62     y2 = round(y1 + Wb);
63     image(x1:x2,y1:y2) = 255;
64     x1 = round(Ys + (steps(1+(n-1)*4+3)-1)*Hb);
65     y1 = round(Xs + (steps(1+(n-1)*4+4)-1)*Wb);
66     x2 = round(x1 + Hb);
67     y2 = round(y1 + Wb);
68     image(x1:x2,y1:y2) = 255;
69     % show removed blocks
70     imshow(image);
71     pause(1);
72
73 end
74 hold off;

```

Listing 3.10: mocklink.m(22-74, visualization)

连线为宽度为 3 像素的**黑线段**，消去的块用白色背景表示。

由于连线可能超出图像块的区域范围，需要特殊处理使得绘制出的连线仍在图片范围内。

最终能够模拟消除能够正常工作，消除过程以动画形式保存为根目录下的[mocklink.gif](#)。

以下取若干帧展示效果：

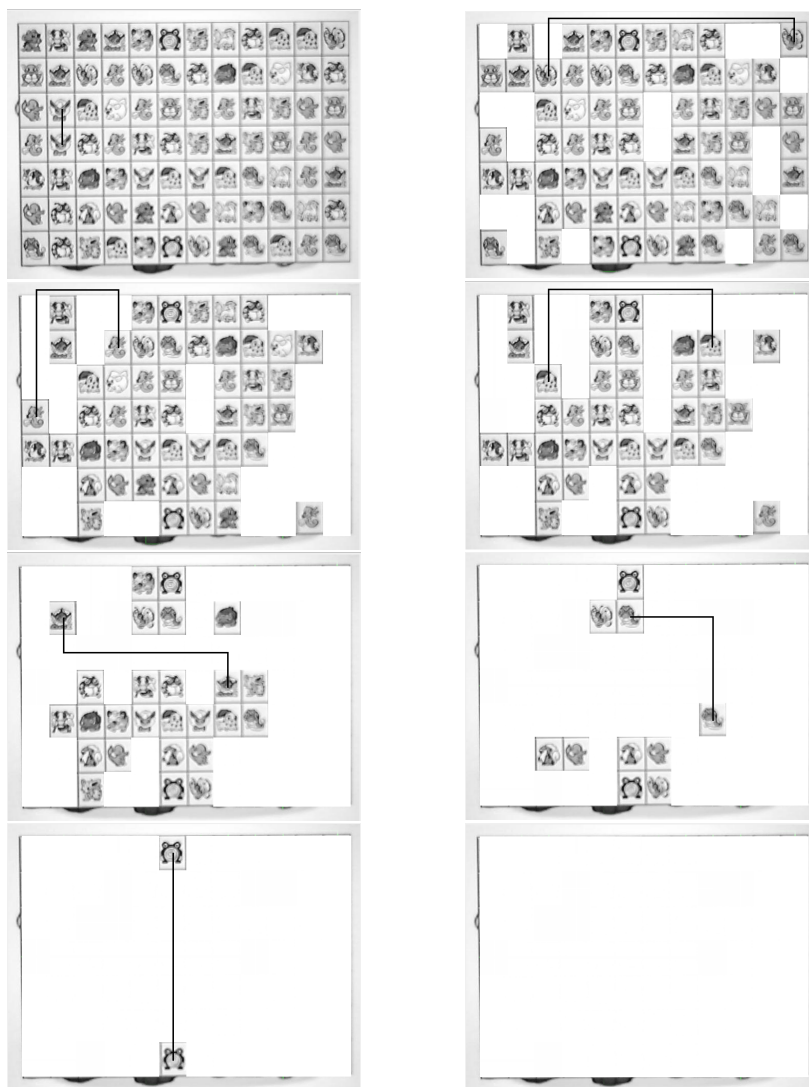


Figure 3.13: 模拟消除过程 (部分)

7. 未能挤出时间完成选做两题 Orz

4 实验总结

- 在判断可连性时，使用画十字的方法确实比较简单易行；
- 采用 `canlink->canlink1->canlink0` 这样类似递归的调用可以提高代码重用率，且具有较低的复杂度；
- 在实现 `omg.m` 时，将人脑的思路用代码实现不失为一种好方法，即先消除相邻的块，再在广义边界上寻找可消除的块，这样可以消除大部分图像块，最后再对剩余的块进行比较暴力的配对；这样比一上来就暴力配对效率高了不少（不过对于图像块较少的情形，效率提升并不明显）；
- 图像分块没用采用傅里叶变换进行分析，而是通过将图像转换为黑白图像增强周期性信息后，再利用 `findpeaks` 函数及其丰富的限制选项对黑白图像的纵横均值进行分析，简单有效；而傅里叶的基频分量很可能淹没在较高能量的噪声分量中；
- 不转为黑白图像直接 `findpeaks` 对 `graygroundtruth` 效果还不错，但处理较模糊且带有些微旋转量的 `graycapture`，转黑白这一步的重要性是不言而喻的；
- 分块完毕后相似度匹配前的高通滤波十分必要，没有做高通滤波的图3.11证明高通滤波有效地提取了各图像块独特的纹理信息；
- 两图像块进行相关运算时（匹配滤波），最好要使图像块各点取值均匀地分布在正负半轴 $(-128,127)$ ，这样计算过程中正负相抵有助于更好地分辨相同块和不同块；若取值范围为 $(0,255)$ 再做相关，那么得到的相似度均偏高，且不能很好地区分相同块和不同块；
- 利用相似度阈值进行聚类是比较自然的思路，事实上 MATLAB 应该自带类似的函数如 `classify` 等；
- 暴力地进行两两相似度运算效率较低，可考虑改进为做相似度运算的同时进行聚类操作，这样避免了重复运算；
- MATLAB 中的矩阵遍历顺序为列优先，预先考虑到这一点可以避免代码中出现不必要的转置（我们主观往往乐于倾向行优先遍历）；
- 可视化过程要考虑画线是否超出边界的较繁琐的问题，表达式较复杂，其实不难；
- 总体而言，连连看大作业还是很有趣；可惜没来得及完成炫酷的选做题；

- 另外，很钦佩 `construct.p` 的原创者，能够构造出几乎保证任意消除顺序均能完全消除的游戏 (观测结果)。