连连看实验报告

郭一隆 (2013011189)

September 5, 2015

Contents

List of Figures

List of Tables

List of Source Codes

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。

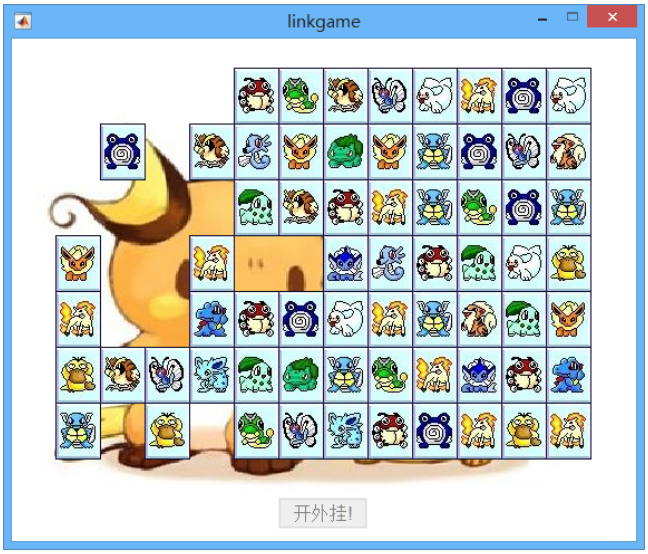

Figure 2.1: 运行连连看

2. 注意 linkgame 目录下有个 detect.p，它的功能时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。

```
function [I,J] = adjcross(mtx,x,y)
% return index vector of can-reach blocks of mtx(x,y) on both directions
% notice: x >= 2 && y >= 2 && all(mtx >= 0)

    % get vectors of four directions
    left = mtx(x,1:y-1);
    right = mtx(x,y+1:end);
    up = mtx(1:x-1,y);
    down = mtx(x+1:end,y);

    % get zero run length adjacent to mtx(x,y)
    lz = find(cumsum(left,'reverse')==0);
    rz = find(cumsum(right,'forward')==0);
    uz = find(cumsum(up,'reverse')==0);
    dz = find(cumsum(down,'forward')==0);

    I = [x*ones(1,length([lz,rz])),uz.',x+dz.'];
    J = [lz,y+rz,y*ones(1,length([uz;dz]))];

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。

```
function bool = canlink0(mtx,x1,y1,x2,y2)
% return 1 if it's a direct link (no turns)

    if x1 == x2 && ~any(mtx(x1,min(y1,y2)+1:max(y1,y2)-1))
        bool = 1;
    elseif y1 == y2 && ~any(mtx(min(x1,x2)+1:max(x1,x2)-1,y1))
        bool = 1;
    else
        bool = 0;
    end

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。

```
function bool = canlink1(mtx,x1,y1,x2,y2)
% return 1 if the turns of link path <= 1

    if canlink0(mtx,x1,y1,x2,y2)
        bool = 1;
        return
    end

    % grow the cross of origin
    [I,J] = adjcross(mtx,x1,y1);

    for n = 1:length(I)
        i = I(n); j = J(n);
        if canlink0(mtx,i,j,x2,y2)
            bool = 1;
            return
        end
    end

    bool = 0;

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。

```
function bool = canlink(mtx,x1,y1,x2,y2)
% return 1 if these two blocks can link!
    if mtx(x1,y1) ~= mtx(x2,y2) || ~mtx(x1,y1) || ~mtx(x2,y2)
        bool = 0;
        return
    end

    if canlink1(mtx,x1,y1,x2,y2)
        bool = 1;
        return
    end

    % grow the cross of origin
    [I,J] = adjcross(mtx,x1,y1);

    for n = 1:length(I)
        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)

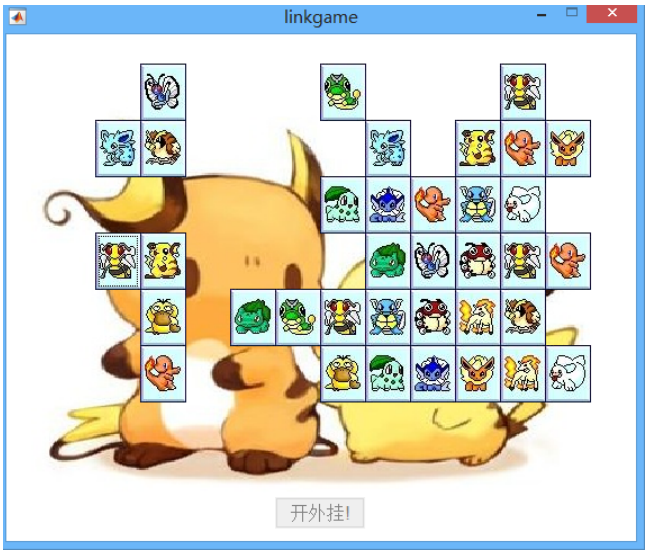

Figure 2.2: 重写 `detect.m` 后运行 `linkgame`

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      rdiff = diff(mtx,1,1);
51      [I,J] = find(rdiff==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)`

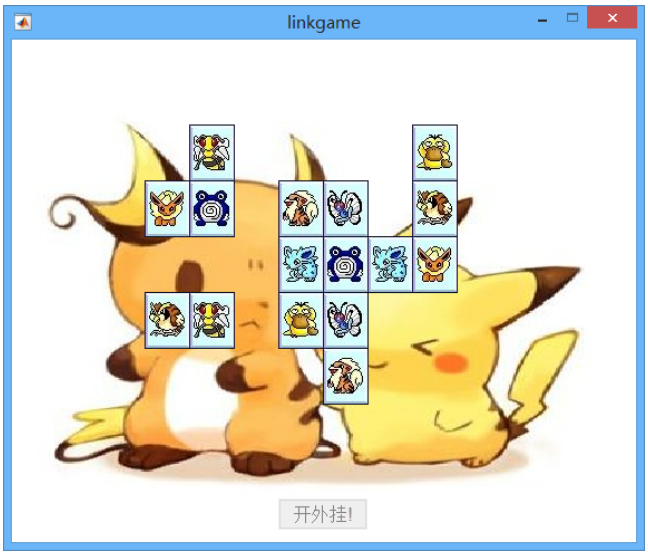

Figure 2.5: 依次调用 `matchadj` 和 `matchborder` 后

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);
```

```
            for m = 1:length(I)
                for n = m+1:length(I)
                    if canlink(mtx,I(m),J(m),I(n),J(n))
                        steps = [steps,I(m),J(m),I(n),J(n)];
                        % update mtx
                        mtx(I(m),J(m)) = 0;
                        mtx(I(n),J(n)) = 0;
                        break
                    end
                end
            end
        end
    end

end
```

Listing 2.8: omg.m(matchrest)

经过多次测试，均能顺利完成功能。

```
function steps = omg(mtx)

    [m,n] = size(mtx);

    % add surrounding zeros
    mtx = [0,zeros(1,n),0;
        zeros(m,1),mtx,zeros(m,1);
        0,zeros(1,n),0];

    [steps1,mtx] = matchadj(mtx);
    [steps2,mtx] = matchborder(mtx);
    [steps3,mtx] = matchrest(mtx);
    steps = [steps1,steps2,steps3];

    % make steps meet interface
    steps = [length(steps)/4,steps-1];

end
```

Listing 2.9: omg.m(main)

3 攻克别人的连连看

1. 在 MALTAB 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：

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

   绘制水平竖直扫描线灰度均值如图3.1。

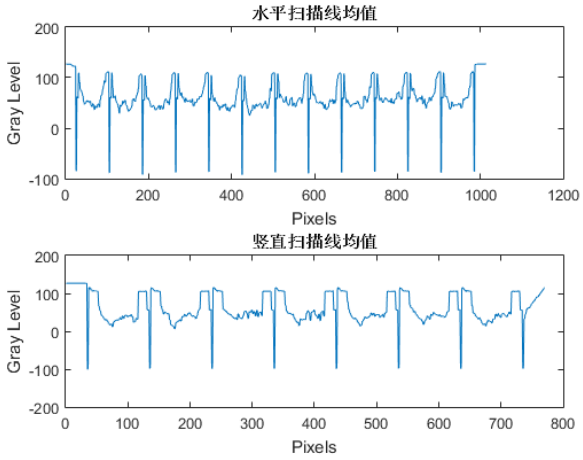

Figure 3.1: 扫描线灰度均值

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

```
%% get block size and position of game region
hor = mean(image,1);
ver = mean(image,2);

npeak = min(hor); % negative peak
I = find(hor<0.9*npeak);
I = sort(I);
Xs = I(1);
I = diff(I); % width between peaks
```

```
I = I(I>30); % filter too small blocks
Wb = mean(I)+1;
Nc = length(I); % number of block columns

npeak = min(ver); % negative peak
I = find(ver<0.9*npeak);
I = sort(I);
Ys = I(1);
I = diff(I); % width between peaks
I = I(I>30); % filter too small blocks
Hb = mean(I)+1;
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：

```
blocks = {};
figure(2);
for i = 1:Nr
    for j = 1:Nc
        block = original(round(Ys+(i-1)*Hb+1):round(Ys+i*Hb),...
            round(Xs+(j-1)*Wb+1):round(Xs+j*Wb));
        blocks{end+1} = block;
        subplot(Nr,Nc,(i-1)*Nc+j);
        imshow(block);
    end
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：

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

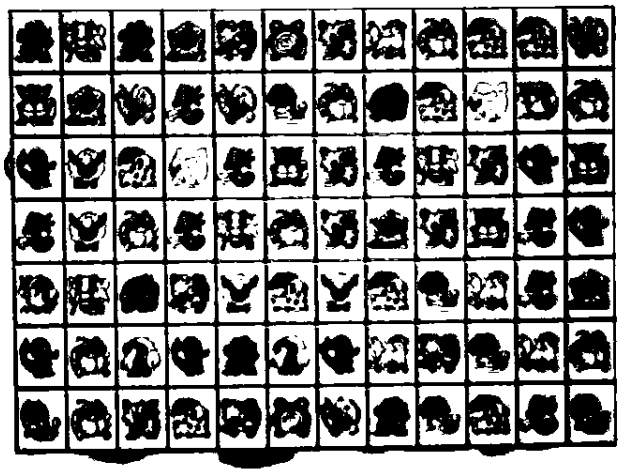

Figure 3.3: `graycapture` 预处理成黑白图像

再观察扫描线均值：

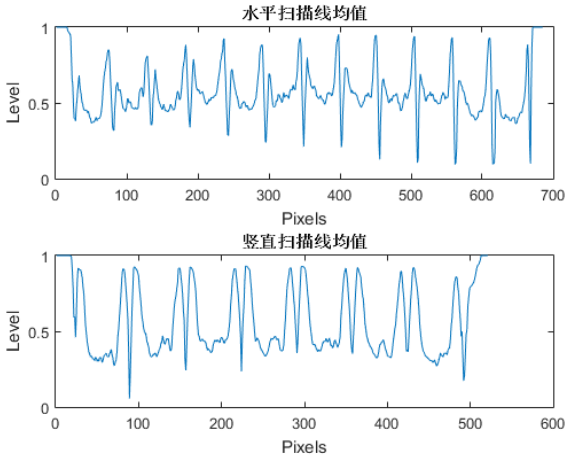

Figure 3.4: 扫描线黑白均值

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: `finkpeaks` 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：

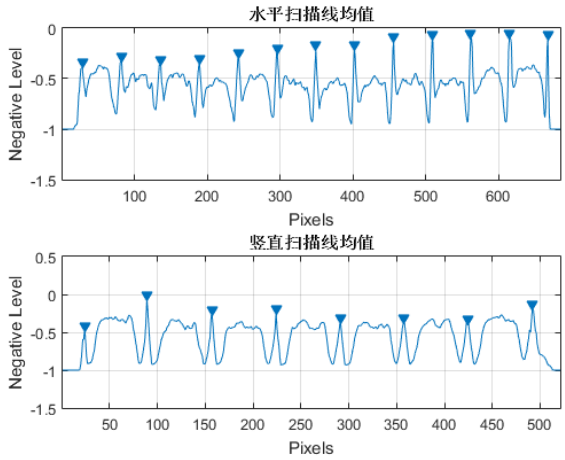

Figure 3.5: `findpeaks` 检测峰值 (纵坐标为原数据的相反数)

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)。

```
%% get block size and position of game region
hor = mean(image,1);
ver = mean(image,2);

[~,I] = findpeaks(-hor,'MinPeakProminence',0.3,'MaxPeakWidth',10,'MinPeakDistance',30);
[~,J] = findpeaks(-ver,'MinPeakProminence',0.3,'MaxPeakWidth',10,'MinPeakDistance',30);
Xs = I(1); Ys = J(1);
Wb = mean(diff(I)); Nc = length(I) - 1;
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 (Xs=25, Ys=35, Wb=80, Hb=100, Nc=12, Nr=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。

```
% ...

hp = [-0.3,-0.3,-0.3;
      -0.3, 0.5,-0.3;
      -0.3,-0.3,-0.3];     % construct high pass filter

N = length(blocks);
for i = 1:N
    im1 = imfilter(double(blocks{i})-128,hp);
    im1 = (im1-min(min(im1)))/(max(max(im1))-min(min(im1)))*255;

% ...
```

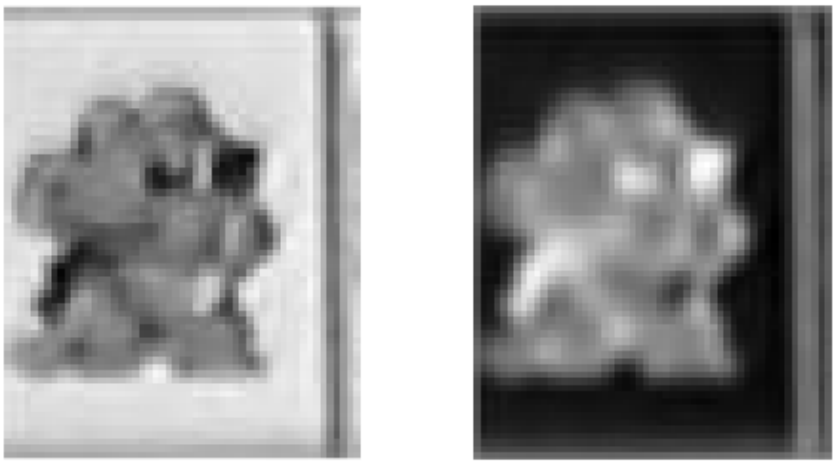

Figure 3.7: 高通滤波提取纹理特征

(b) **计算两图像相似度**：匹配滤波。

```
function S = similarity(im1,im2)
% return similarity of two intensity images: im1 and im2
% notice: 0 <= S <= 1

    im1 = double(im1);
    im2 = double(im2);
    S = max(max(filter2(im1,im2)) / ...
        sqrt(sum(sum(im1.^2))*sum(sum(im2.^2))));

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。

```
function match_map = similarsort(blocks)
% <1-by-n cell> blocks
% <m-by-3 matrix> match_map: m == n*(n-1)/2,
%       the first 2 elements are indexes of blocks,
%       the third element is similarity
%       rows of match_map are sorted by similarity

    match_map = [];

```

```
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)，这样在做相似度运算时，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)

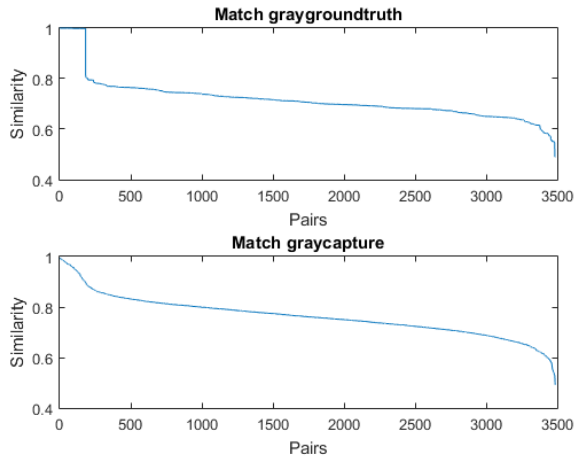

Figure 3.8: 相似度走势曲线

- 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：

```
figure;

for i = 1:10
    subplot('position',[0.1+(i-1)*0.08,0.55,0.08,0.1]);
    imshow(blocks{match_map2(i,1)}); title(num2str(match_map2(i,3)));
    subplot('position',[0.1+(i-1)*0.08,0.35,0.08,0.1]);
    imshow(blocks{match_map2(i,2)});
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：

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

```
figure;
count = 0;
kindt = kind.';

for i = 1:size(match_map2,1)
    if kindt(match_map2(i,1)) ~= kindt(match_map2(i,2))
        count = count + 1;
        subplot('position',[0.1+(count-1)*0.08,0.55,0.08,0.1]);
        imshow(blocks{match_map2(i,1)}); title(num2str(match_map2(i,3)));
        subplot('position',[0.1+(count-1)*0.08,0.35,0.08,0.1]);
        imshow(blocks{match_map2(i,2)});
        xlabel(['index:',num2str(i)]);
        if count == 10
            break
        end
    end
end
```

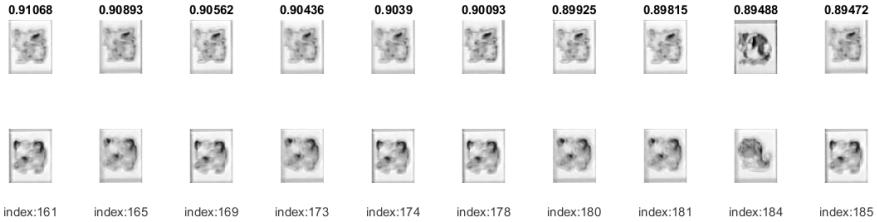

Figure 3.10: 不同块最相似前十名

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`：

```
function category = classify(match_map,threshold,N)
% <m-by-3 matrix> match_map: match_map(i,3) = similarity
% <int> N: number of elements to be classified
% classify by similarity
% they're in the same class if similarity > threshold

% <n-by-1 cell> category: n CLASSes

    category{1} = match_map(1,1:2);

    isclassified = zeros(1,N);  % mark CLASS of each element
    isclassified(match_map(1,1:2)) = 1;     % CLASS 1

    for i = 1:size(match_map,1)
        row = match_map(i,:);
        if row(3) < threshold
            indexes = find(isclassified==0);
            for index = indexes
                [J,I] = find(match_map.'==index,10);   % find first 10 pairs containing INDEX
                inds = sub2ind(size(match_map),I,3-J);
                class = mode(isclassified(match_map(inds)));
                category{class} = union(category{class},match_map(I(1),J(1)));
                isclassified(match_map(I(1),J(1))) = class;
            end
            % finished
            break
        end

        status = isclassified(row(1:2));    % classify status
        if sum(status) == 0
            % all not classified
            % new class
            category{end+1} = row(1:2);
            isclassified(row(1:2)) = length(category);
        elseif all(status~=0)
            % all classified
            class1 = isclassified(row(1));
            class2 = isclassified(row(2));
            if class1 ~= class2
                % merge class2 into class1
                e = category{class2};
```

```
                category{class1} = [category{class1},e];
                isclassified(e) = class1;
                category{class2} = [];
            end
        else
            % one classified, the other NOT
            if status(1) == 0
                class = isclassified(row(2));
                category{class} = [category{class},row(1)];
                isclassified(row(1)) = class;
            else
                class = isclassified(row(1));
                category{class} = [category{class},row(2)];
                isclassified(row(2)) = class;
            end
        end
    end

    % delete empty cells
    result = {};
    for i = 1:length(category)
        if ~isempty(category{i})
            result{end+1} = category{i};
        end
    end

    category = result;

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：

```
>> category1 = classify(match_map1,0.9,84);
>> 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：

```
figure;

for i = 1:length(category2)
    subplot(5,5,i);
    C = category2{i};
    imshow(blocks{C(1)});
    title(['#',num2str(i),' Counts: ',num2str(length(C))]);
end
```

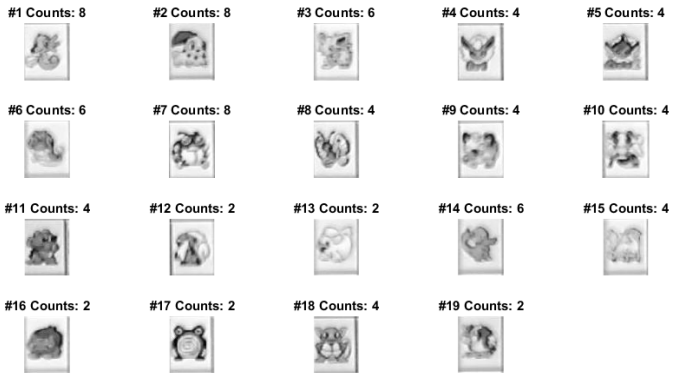

Figure 3.12: 对 graycapture 自动分类结果

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：

```
mtx = zeros(Nc,Nr);

for i = 1:length(category2)
    mtx(category2{i}) = i;  % rowwise index
end

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。