连连看实验报告

郭一隆 (2013011189)

September 4, 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。

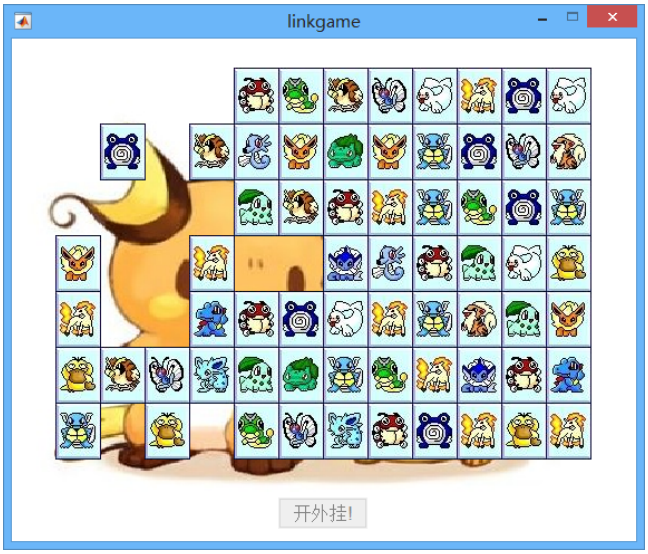

Figure 2.1: 运行连连看

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)

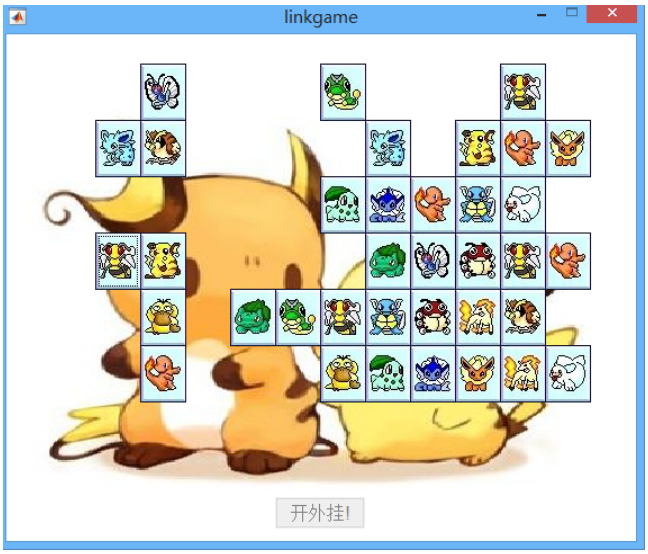

Figure 2.2: 重写 `detect.m` 后运行 `linkgame`

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)`

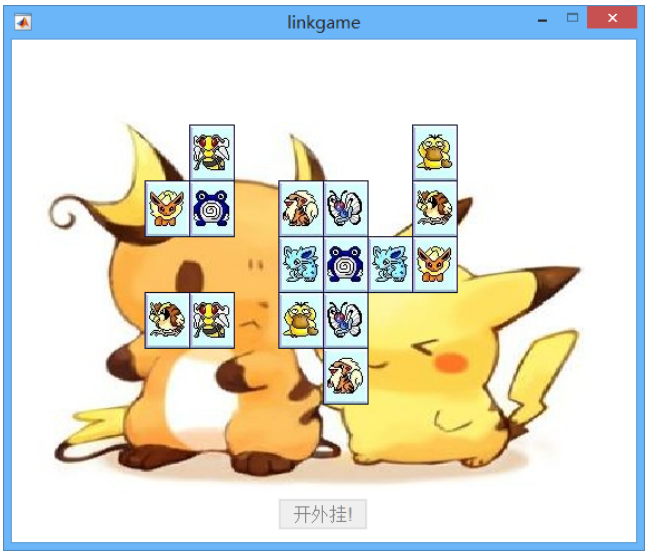

Figure 2.5: 依次调用 `matchadj` 和 `matchborder` 后

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。

   绘制水平竖直扫描线灰度均值如图3.1。

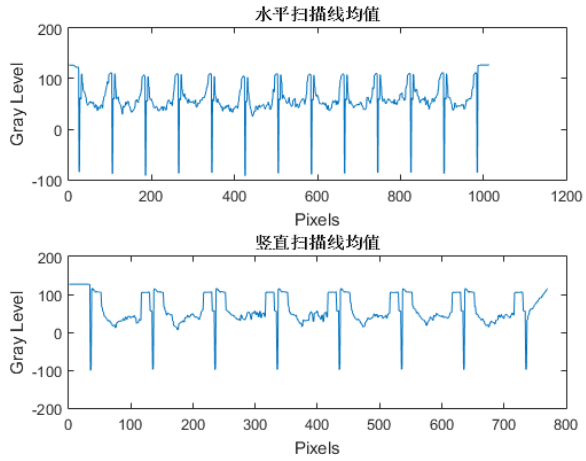

Figure 3.1: 扫描线灰度均值

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);

figure(1);
subplot 211; plot(hor);
xlabel('Pixels'); ylabel('Gray Level'); title('    ';(
subplot 212; plot(ver);
xlabel('Pixels'); ylabel('Gray Level'); title('    ';(

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

然后按尺寸获取各图像块，效果良好：

```
%% get all blocks
blocks = {};
figure(2);
for i = 1:Nr
    for j = 1:Nc
        block = image(Ys+(i-1)*Hb+1:Ys+i*Hb,Xs+(j-1)*Wb+1:Xs+j*Wb);
        blocks{end+1} = block;
        subplot(Nr,Nc,(i-1)*Nc+j);
        imshow(uint8(block+128));
    end
end
```

Listing 3.2: `divide.m` 获取所有块

Figure 3.2: 图像分割效果