

```
1  foreach $key (keys %delete_list) {
2    my $dir = $build_photo_path;
3    my (@files) = ();
4    &header;
5    opendir(DIR, $dir) || &err("can't open : $!");
6    @files = grep { /$key\./i } readdir(DIR);
7    closedir(DIR);
8    if ($#files > -1) {
9      foreach (@files) {
10         unlink("$dir/$_") || &err("can't delete $_ : $!");
11      }
12    }
13 }
```

```
opendir(DIR, $build_photo_path) || &err("can't open $build_photo_path: $!")  
;  
@all_files = readdir(DIR);  
closedir(DIR);  
  
foreach $key (keys %delete_list) {  
    &header;  
    foreach (grep { /^$key\./i } @all_files) {  
        unlink("$build_photo_path/$_") || &err("can't delete $_ : $!");  
    }  
}
```

```

1  #!/usr/local/bin/perl5 -w
2
3  $input1 = 'chaintest.scan';
4  $output= "OUT";
5  $output2= "OUT.out";
6
7  open (INFILE,$input1)||die "cannot open $input1";
8  open(OUTFILE,">$output")||die "canoot\n";
9  open(OUTFILE2,">$output2")||die "canoot\n";
10
11 #####
12     &preprocess;
13 open (INFILE2,$output)||die "cannot open $input1";
14     $/="";
15     $ct_scanout = 0;
16     while (<INFILE2>){
17         chomp;
18         $ct_scanout = 1 if (/apply\s*"grp[0-9]_unload"/);
19         $chain_test=1 if (/CHAIN_TEST/);
20
21 if (( /\t*chain\s+"chain([0-9])\"/) && ($chain_test)){
22     $chain_number = $1;
23     &cleanup;
24     $chain_input = (split /=/,$_)[1];
25     $chain_input =~ tr/\//d;
26     SWITCH: {
27         $chain_number==1 && do {
28             @chain1 = split (//,$chain_input);
29             $scan_chain_length1 = @chain1; };
30         $chain_number==2 && do {
31             @chain2 = split (//,$chain_input);
32             $scan_chain_length2 = @chain2; };
33         $chain_number==3 && do {
34             @chain3 = split (//,$chain_input);
35             $scan_chain_length3 = @chain3; };
36         $chain_number==4 && do {
37             @chain4 = split (//,$chain_input);
38             $scan_chain_length4 = @chain4; };
39         $chain_number==5 && do {
40             @chain5 = split (//,$chain_input);
41             $scan_chain_length5 = @chain5; };
42         $chain_number==6 && do {
43             @chain6 = split (//,$chain_input);
44             $scan_chain_length6 = @chain6; };
45         $chain_number==7 && do {
46             @chain7 = split (//,$chain_input);
47             $scan_chain_length7 = @chain7;
48             &printout;
49             };
50     }#END SWITCH
51     }
52 } #end of While statement
53 sub printout {
54     if ($ct_scanout ){
55         for ($i=0;$i<$scan_chain_length3;$i++){
56             $chain1[$i] =~ s/0/L/g; $chain1[$i] =~ s/1/H/g;
57             $chain2[$i] =~ s/0/L/g; $chain2[$i] =~ s/1/H/g;
58             $chain3[$i] =~ s/0/L/g; $chain3[$i] =~ s/1/H/g;
59             $chain4[$i] =~ s/0/L/g; $chain4[$i] =~ s/1/H/g;
60             $chain5[$i] =~ s/0/L/g; $chain5[$i] =~ s/1/H/g;
61             $chain6[$i] =~ s/0/L/g; $chain6[$i] =~ s/1/H/g;
62             $chain7[$i] =~ s/0/L/g; $chain7[$i] =~ s/1/H/g;
63             print OUTFILE2 "\n(ct_so
64 $chain1[$i]$chain2[$i]$chain3[$i]$chain4[$i]$chain5[$i]$chain6[$i]$
chain7[$i]
65     )";

```

```
66                                     }
67         $ct_scanout=0;
68     }
69     elsif ($ct_scanout==0){
70         for ($i=0;$i<$scan_chain_length3;$i++){
71             $chain1[$i] =~ s/X/0/g; $chain2[$i] =~ s/X/0/g;
72             $chain3[$i] =~ s/X/0/g; $chain4[$i] =~ s/X/0/g;
73             $chain5[$i] =~ s/X/0/g; $chain6[$i] =~ s/X/0/g;
74             $chain7[$i] =~ s/X/0/g;
75             print OUTFILE2 "\n(ct_si $ct_si{tdi}
76 $chain1[$i]$chain2[$i]$chain3[$i]$chain4[$i]$chain5[$i]$chain6[$i]$
chain7[$i]
77 )";
78         }
79     }
80 }
81 sub cleanup{
82     s/^\s+|\s*\n+$/g;
83     tr/\t //d;
84     s/\n//g;
85     s/\\//g;
86 }
87 sub preprocess{
88     while (<INFILE>){
89         chomp;
90         s/$//g if (/apply/);
91         print OUTFILE ("$_\n");
92         last if ( /^SCAN_CELLS/);
93     }
94     close OUTFILE;
95 }
```

```

1  #!/usr/local/bin/perl5 -w
2
3  $input1 = 'chaintest.scan';
4  $output= "OUT";
5  $output2= "OUT.out";
6
7  open (INFILE,"$input1")||die "cannot open $input1";
8  open(OUTFILE,">$output")||die "canoot\n";
9  open(OUTFILE2,">$output2")||die "canoot\n";
10
11 #####
12 &preprocess;
13 open (INFILE2,"$output")||die "cannot open $input1";
14 $/= "";
15 $ct_scanout = 0;
16 while (<INFILE2>) {
17     chomp;
18     $ct_scanout = 1 if (/apply\s*"grp[0-9]_unload"/);
19     $chain_test=1 if (/CHAIN_TEST/);
20
21     if ((/\t*chain\s+"chain([0-9])"/) && ($chain_test)){
22         $chain_number = $1;
23         &cleanup;
24         $chain_input = (split /=/, $_)[1];
25         $chain_input =~ tr/"//d;
26         $chain[$chain_number] = [split //, $chain_input];
27         &printout if $chain_number == 7;
28     }
29 }
30
31 sub printout {
32     if ($ct_scanout ){
33         for ($i=0;$i<@{$chain[3]};$i++){
34             for (@chain) {
35                 $_->[$i] =~ s/0/L/g;
36                 $_->[$i] =~ s/1/H/g;
37             }
38             my @chars = map $_->[$i], @chain;
39             print OUTFILE2 "\n(ct_so\n", join("", @chars), "\n )";
40         }
41         $ct_scanout=0;
42     } elsif ($ct_scanout==0) {
43         for ($i=0;$i<@{$chain[3]};$i++) {
44             for (@chain) {
45                 $_->[$i] =~ s/X/0/g;
46             }
47             my @chars = map $_->[$i], @chain;
48             print OUTFILE2 "\n(ct_si $ct_si{tdi}\n", join("", @chars), "
\n )";
49         }
50     }
51 }
52
53 sub cleanup {
54     s/^\s+|\s*\n+$/g;
55     tr/\t //d;
56     s/\n//g;
57     s/\\//g;
58 }
59
60 sub preprocess {
61     while (<INFILE>){
62         chomp;
63         s/$/;/g if (/apply/);
64         print OUTFILE ("$_\n");
65         last if ( /^SCAN_CELLS/);

```

```
66     }  
67     close OUTFILE;  
68 }
```

```

1  #!/usr/local/bin/perl5 -w
2
3  $input = 'chaintest.scan';
4  $output= "OUT.out";
5
6  open (INFILE,$input)||die "cannot open $input";
7  open(OUTFILE,">$output")||die "canoot\n";
8
9  #####
10 $/= " ";
11 $ct_scanout = 0;
12
13 while (<INFILE>) {
14     chomp;
15     last if /^SCAN_CELLS/;
16     $seen_CHAIN_TEST=1 if /CHAIN_TEST/;
17     next unless /apply/ && $seen_CHAIN_TEST;
18     die "Missing ';' in 'apply' line" if /apply[^\;]*\n/;
19
20     $ct_scanout = 1 if /apply\s*"grp[0-9]_unload"/;
21     next unless /\t*chain\s+"chain([0-9])"/;
22     $chain[$1] = cleanup($_);
23     &printout if $1 == 7;
24 }
25
26 sub printout {
27     if ($ct_scanout ) {
28         for ($i=0;$i<@{$chain[3]};$i++){
29             for (@chain) {
30                 next unless defined;
31                 $_->[$i] =~ s/0/L/g;
32                 $_->[$i] =~ s/1/H/g;
33             }
34             my @chars = map $_->[$i], @chain;
35             print OUTFILE "\n(ct_so\n", join("", @chars), "\n )";
36         }
37         $ct_scanout=0;
38     } elsif ($ct_scanout==0) {
39         for ($i=0;$i<@{$chain[3]};$i++) {
40             for (@chain) {
41                 next unless defined;
42                 $_->[$i] =~ s/X/0/g;
43             }
44             my @chars = map $_->[$i], @chain;
45             print OUTFILE "\n(ct_si  $ct_si{tdi}\n", join("", @chars), "\n )";
46         }
47     }
48 }
49
50 sub cleanup {
51     local $_ = shift;
52     s/.*=//s;
53     tr/01X//cd;
54     return [ split // ]
55 }
56

```

```
#!/usr/local/bin/perl5 -w

$input = 'chaintest.scan';
$output = "OUT.out";

open (INFILE, $input) || die "cannot open $input: $!";
open (OUTFILE, "> $output") || die "cannot open $output: $!";

$/="";

while (<INFILE>) {
    chomp;
    last if /^SCAN_CELLS/;
    $seen_CHAIN_TEST=1 if /CHAIN_TEST/;
    next unless /apply/ && $seen_CHAIN_TEST;
    die "Missing ';' in 'apply' line" if /apply[^\;]*\n/;

    my $ct_scanout = /apply\s*"grp[0-9]_unload"/;
    next unless /\t*chain\s+"chain([0-9])"/;
    $chain[$1] = cleanup($_);
    if ($1 == 7) {
        printout($ct_scanout, @chain) ;
        @chain = ();
    }
}

sub printout {
    my ($scanout, @chain) = @_;
    for my $i (0 .. $#{$chain[1]}){
        my $chars = "";
        for (@chain) {
            next unless defined;
            $chars .= $_->[$i];
        }
        if ($scanout) {
            $chars =~ tr/01/LH/;
        } else {
            $chars =~ tr/X/0/;
        }
        print OUTFILE
            $scanout ? "\n(ct_so\n" : "\n(ct_si  $ct_si{tdi}\n",
            "$chars\n)";
    }
}

sub cleanup {
    local $_ = shift;
    s/.*=//s;
    tr/01X//cd;
    return [ split // ]
}

```

```
1    #!/usr/bin/perl
2    #
3
4    #
5    #A very basic tic-tac-toe program (the computer chooses randomly
)
6    #
7
8
9    use strict;
10   use CGI;
11   use Socket;
12
13   my ($rounds, $round_temp, $squares, $page, $x, $y, $z, $cell, $p
layer_move, @available_choices, $computer_move, @choices, $round, $winner, $
player_move_pretty, $computer_move_pretty);
14   my ($round_minus_one); #bug fix (was recording moves for round1
as round2
15
16   $page = CGI->new();
17
18   print $page->header;
19   print $page->start_html();
20
21   # print table beginnings
22
23   print("<table width=\"90%\" border=0 cellpadding=15>\n");
24   print("<tr valign=middle>\n");
25
26   # left cell is tic tac toe table
27
28   print("<td align=center>\n");
29
30   #
31   # find out which round it is so we know how to define $squares
32   #
33
34   unless ($page->param('round')) {
35       $round = 0;
36   }else {
37       $round = $page->param('round');
38   }
39
40   #
41   # set array of tic tac toe squares
42   #
43
44   if ($round > 0) {
45       $squares = [
46           [
47               $page->param('[0][0]'),
48               $page->param('[0][1]'),
49               $page->param('[0][2]')
50           ],
51           [
52               $page->param('[1][0]'),
53               $page->param('[1][1]'),
54               $page->param('[1][2]')
55           ],
56           [
57               $page->param('[2][0]'),
58               $page->param('[2][1]'),
59               $page->param('[2][2]')
60           ]
61       ];
62   }else {
```

```
63     $squares = [
64         ['?', '?', '?'],
65         ['?', '?', '?'],
66         ['?', '?', '?']
67     ];
68 }
69
70 #
71 # set array for determining history of moves (recorded by round)
72 #
73
74 if ($round > 0) {
75     $rounds = {
76         round1 => {
77             player => $page->param('round1_x'),
78             computer => $page->param('round1_o')
79         },
80         round2 => {
81             player => $page->param('round2_x'),
82             computer => $page->param('round2_o')
83         },
84         round3 => {
85             player => $page->param('round3_x'),
86             computer => $page->param('round3_o')
87         },
88         round4 => {
89             player => $page->param('round4_x'),
90             computer => $page->param('round4_o')
91         },
92         round5 => {
93             player => $page->param('round5_x'),
94             computer => $page->param('round5_o')
95         }
96     };
97 }else {
98     $rounds = {
99         round1 => {
100             player => '?',
101             computer => '?'
102         },
103         round2 => {
104             player => '?',
105             computer => '?'
106         },
107         round3 => {
108             player => '?',
109             computer => '?'
110         },
111         round4 => {
112             player => '?',
113             computer => '?'
114         },
115         round5 => {
116             player => '?',
117             computer => '?'
118         }
119     };
120 }
121
122 #
123 # increment $round to give it a new hidden value
124 #
125
126 $round = $round + 1;
127 $round_minus_one = $round - 1;
128 print ("Round is: $round<br>\n");
```

```

129
130     ##
131     ## get player move using subroutine (subroutine stores it to $sq
uares array)
132     ##
133
134     $player_move = $page->param('choice');
135     if ($player_move) {
136         $round_temp = "round" . $round_minus_one;
137         player_moves($player_move, $page, $squares);
138         $player_move_pretty = make_move_pretty($player_move);
139         $rounds->{$round_temp}->{'player'} = $player_move_pretty;
140
141         #
142         # evaluate for winner after player moves
143         #
144
145         $winner = evaluate_board($squares);
146         if ($winner eq "x") {
147             print ("<font color='blue'>Player Won!</font><p>\n");
148             print_final_table($squares, $page, $winner, $round, $round
s);
149             }else {
150
151             #
152             # check to see if player won. if player won, don't do the rest o
f this
153
154             #
155             # get available choices for computer choices
156             #
157
158             @available_choices = get_available_choices($squares);
159
160             #
161             # get computer move
162             #
163
164             $computer_move = get_computer_choice(@available_choices);
165             ($x, $y) = split(/:/, $computer_move);
166             #get coordinates for computer move and store in $x and
$y
167             $squares->[$x][$y] = "o";    ## change square to "o"
168             $round_temp = "round" . $round_minus_one;
169             $computer_move_pretty = make_move_pretty($computer_move);
170             $rounds->{$round_temp}->{'computer'} = $computer_move_pret
ty;
171
172             } #matches else {
173
174             } #matches if ($player_move)
175
176             #
177             # now that we have the array with computer and player choices, s
ee if there is a winner
178             #
179             $winner = evaluate_board($squares);
180             if ($winner eq "o") { #we already checked for x before
181                 print ("<font color = 'blue'>Computer Won!</font><p>\n");
182                 print_final_table($squares, $page, $winner, $round, $rounds);
183             }
184
185             if (($winner ne "o") and ($winner ne "x")) {
186                 print ("<font color='blue'>No winner yet</font><p>\n");
187
188

```

```

189     # start form
190
191     print $page->startform(-method=> 'POST');
192
193     #print hidden values in form
194
195     print_hidden_values($page, $squares, $rounds);
196
197     # print hidden value for $round
198
199     print "<input type='hidden' name = 'round' value='$round'>\n"
;
200
201     # print hidden values for saving rounds
202
203
204     # start tic tac toe table
205
206     print ("<table border=1 cellpadding=10>\n<tr>");
207
208     #
209     # look through array elements ($squares) and find x's or o's
210     # for all squares with no value, print a checkbox with the coord
inates
211     # in the form of [0][0] as its name
212     #
213
214     foreach $x(0..2) {
215         if ($squares->[0][$x] eq "?") {
216             print_cell($squares, $page, $x, "0");
217         }else {
218             print ("<td>" . $squares->[0][$x] . "</td>\n");
219         }
220     }
221
222     print ("</tr><tr>\n");
223     foreach $x(0..2) {
224         if ($squares->[1][$x] eq "?") {
225             print_cell($squares, $page, $x, "1");
226         }else {
227             print ("<td>" . $squares->[1][$x] . "</td>\n");
228         }
229     }
230
231     print ("</tr><tr>\n");
232     foreach $x(0..2) {
233         if ($squares->[2][$x] eq "?") {
234             print_cell($squares, $page, $x, "2");
235         }else {
236             print ("<td>" . $squares->[2][$x] . "</td>\n");
237         }
238     }
239
240     print ("</tr></table>");
241
242     #
243     # print warning about picking multiple squares
244     #
245
246     print $page->submit();
247
248     print ("<p><font color='red'>Note: if you pick more than one
square, your choice will be the upper and leftmost square that you choose!!<
/font><p>\n");
249
250

```

```

251     print $page->endform();
252     }
253
254     # end table cell
255
256     print ("</td>\n");
257     print ("<td align=middle>\n");
258
259     # get printable versions of moves and print choices
260
261     if (($player_move) or ($computer_move)) {
262         foreach $x(1..$round_minus_one) {
263             $round_temp = "round" . $x;
264             print ("<p><b>Round $x:</b><br>\n");
265             print ("\tplayer: " . $rounds->{$round_temp}->{'player'} .
" <br>\n");
266             print ("\tcomputer: " . $rounds->{$round_temp}->{'computer
'} . " <br>\n");
267         }
268     }else {
269         print ("No moves yet.\n");
270     }
271
272     # end table
273
274     print ("</td></tr></table>\n");
275
276     print $page->end_html();
277
278
279     sub player_moves {
280         my $move = $_[0]; ##get move
281         my $page = $_[1]; ## import page object
282         my $squares = $_[2]; ## import array so we can change square
283         my ($x, $y, $z);
284
285         foreach $x(0..2) {
286             foreach $y(0..2) { ##test for each square
287                 $z = "$x[$y]";
288                 if ($move eq $z) {
289                     $squares->[$x][$y] = "x"; ## define array element
for choice
290                 }
291             }
292         }
293     }
294
295     sub print_cell {
296         my $squares = $_[0]; ##import semi-existant array of squares
297         my $page = $_[1]; ##import html stuffs
298         my $x = $_[2]; ##import the number that the foreach is
on
299         my $row = $_[3]; ##import the row we are on
300         my $cell = "$row[$x]"; ##get square coordinates for use in
naming checkbx
301
302         print ("<td>");
303         print ("<input type='checkbox' name='choice' value='$cell'>\n
");
304         print ("</td>");
305     }
306
307     sub get_available_choices {
308         my $squares = $_[0];
309         my ($x, $y, $z);
310         my @available_choices = ();

```

```

311
312     foreach $x(0..2) {
313         foreach $y(0..2) {
314             unless (($squares->[$x][$y] eq "x") or ($squares->[$x][
$y] eq "o")) {
315                 $z = "$x:$y";    ## return in this form for later u
se (so we can split it by the colon)
316                 push (@available_choices, $z);
317             }
318         }
319     }
320     return @available_choices;
321 }
322
323 sub get_computer_choice {
324     my @available_choices = @_;
325     my $length = @available_choices;
326     my $number = int(rand() * ($length - 1));
327     my $choice = $available_choices[$number];
328     return $choice; ## this will be a coordinate, in the form of
$x:$y from $z above
329 }
330
331
332 sub print_hidden_values {
333     my $page = $_[0];
334     my $squares = $_[1];
335     my $rounds = $_[2];
336     my ($x, $y, $cell, $round);
337
338     #print hidden values for cells
339     foreach $x(0..2) {
340         foreach $y(0..2) {
341             $cell = "$x[$y]";
342             print("<input type='hidden' name='$cell' value='" .
$squares->[$x][$y] . "'>");
343             print("\n");
344         }
345     }
346
347     #print hidden values for rounds (history)
348     foreach $x(1..5) {
349         $round = "round" . $x;
350         print("<input type='hidden' name='$round" . "_x' value ='"
. $rounds->{$round}->{'player'} . "'>\n");
351         print("<input type='hidden' name='$round" . "_o' value='"
. $rounds->{$round}->{'computer'} . "'>\n");
352     }
353 }
354
355
356 sub evaluate_board {
357     my ($squares) = $_[0];
358     my ($x, $y, $winner);
359
360     foreach $x (0..2) {
361         if (
362             ($squares->[$x][0] eq $squares->[$x][1]) and
363             ($squares->[$x][1] eq $squares->[$x][2])
364         ) {
365             $winner = $squares->[$x][0];
366             return $winner;
367         }
368     }
369     foreach $y (0..2) {
370         if (

```

```

371             ($squares->[0][$y] eq $squares->[1][$y]) and
372             ($squares->[1][$y] eq $squares->[2][$y])
373             ) {
374                 $winner = $squares->[0][$y];
375                 return $winner;
376             }
377         }
378     if (
379         (($squares->[1][1] eq "x") or ($squares->[1][1] eq "o"))
380         and
381         (
382             (($squares->[0][0] eq $squares->[1][1]) and
383             ($squares->[1][1] eq $squares->[2][2]))
384             or
385             (($squares->[0][2] eq $squares->[1][1]) and
386             ($squares->[1][1] eq $squares->[2][0]))
387         )
388         ) {
389             $winner = $squares->[1][1];
390             return $winner;
391         }
392     }
393
394     sub print_final_table {
395         my $squares = $_[0];
396         my $page = $_[1];
397         my $winner = $_[2];
398         my $round = $_[3];
399         my $rounds = $_[4];
400         my ($visitor, $visitor_name, $time);
401
402         ## print ending table
403
404         print $page->startform(action=>'tic_tac.cgi',
405                               method=>'POST'
406                               );
407
408         print_hidden_values($page,$squares,$rounds);
409
410         print ("<input type='hidden' name='round', value='$round'>\n"
411 );
412         print ("<table border=1 cellpadding=10>\n");
413         print ("<tr valign=middle>\n");
414         foreach $x(0..2) {
415             ##print first row
416             print ("<td align=center>" . $squares->[0][$x] . "</td>\n"
417 );
418         }
419         print ("</tr><tr valign=middle>\n");
420         foreach $x(0..2) {
421             ##print second row
422             print ("<td align=center>" . $squares->[1][$x] . "</td>\n"
423 );
424         }
425         print ("</tr><tr valign=middle>\n");
426         foreach $x(0..2) {
427             ##print third row
428             print ("<td align=center>" . $squares->[2][$x] . "</td>\n"
429 );
430         }
431         print ("</tr></table><p>\n");
432
433         print ("<a href=\"http://soya.serve.com/cgi-bin/tic_tac.cgi\"
434 >Play Again!</a>");
435
436         print $page->end_html();
437     }

```

```
431     #print log of play
432
433     $visitor = $page->remote_host();
434     if ($visitor =~ /\d*\.\d*\.\d*\.\d*/) {
435         $visitor_name = gethostbyaddr(inet_aton($visitor), AF_INET);
436     }
437
438     $time = localtime(time());
439
440     open (MAIL, "| /usr/sbin/sendmail -t");
441     print MAIL "To: author@example.com\n";
442     print MAIL "Subject: tic tac toe results\n";
443     print MAIL "\n$visitor, $visitor_name: $time: $winner on roun
d $round";
444     close MAIL;
445 }
446
447 sub make_move_pretty {
448     my ($move) = $_[0]; #get move (either $player_move or $compu
ter_move
449     my (%squares_names, $pretty_move);
450
451     if ($move =~ /:/) {
452         $move =~ s/^\(\d\):(\d)/$1$2/;
453     }
454     else {
455         $move =~ s/^\[(\d)\]\[(\d)\]/$1$2/;
456     }
457
458     %squares_names = ("00" => "top left",
459                      "01" => "top center",
460                      "02" => "top right",
461                      "10" => "center left",
462                      "11" => "center",
463                      "12" => "center right",
464                      "20" => "lower left",
465                      "21" => "lower center",
466                      "22" => "lower right"
467                      );
468
469     $pretty_move = $squares_names{$move};
470     return $pretty_move;
471 }
472
473
```

```
1  #!/usr/bin/perl
2  #
3
4  #
5  #A very basic tic-tac-toe program (the computer chooses randomly)
6  #
7
8
9  use strict;
10 use CGI;
11 use Socket;
12
13 my ($rounds, $round_temp, $squares, $page, $x, $y, $z, $cell,
14     $player_move, @available_choices, $computer_move, @choices, $round,
15     $winner, $player_move_pretty, $computer_move_pretty);
16
17 my ($round_minus_one); #bug fix (was recording moves for round1 as round
d2 18
19     $page = CGI->new();
20
21     print $page->header;
22     print $page->start_html();
23
24     # print table beginnings
25
26     print("<table width=\"90%\" border=0 cellpadding=15>\n");
27     print("<tr valign=middle>\n");
28
29     # left cell is tic tac toe table
30
31     print("<td align=center>\n");
32
33     #
34     # find out which round it is so we know how to define $squares
35     #
36
37     unless ($page->param('round')) {
38         $round = 0;
39     } else {
40         $round = $page->param('round');
41     }
42
43     #
44     # set array of tic tac toe squares
45     #
46
47     for my $x (0..2) {
48         for my $y (0..2) {
49             $squares->[$x][$y] = $page->param("$x$y");
50         }
51     }
52
53     #
54     # set array for determining history of moves (recorded by round)
55     #
56
57     if ($round > 0) {
58         for my $rn (1..5) {
59             $rounds->{"round$rn"} =
60                 { player => $page->param("round${rn}_x"),
61                   computer => $page->param("round${rn}_o"),
62                 }
63         }
64     }
65
```

```

66     #
67     # increment $round to give it a new hidden value
68     #
69
70     $round = $round + 1;
71     $round_minus_one = $round - 1;
72     print ("Round is: $round<br>\n");
73
74     ##
75     ## get player move using subroutine (subroutine stores it to $squares ar
ray)
76     ##
77
78     $player_move = $page->param('choice');
79     if ($player_move) {
80         $round_temp = "round" . $round_minus_one;
81         player_moves($player_move, $page, $squares);
82         $player_move_pretty = make_move_pretty($player_move);
83         $rounds->{$round_temp}->{'player'} = $player_move_pretty;
84
85     #
86     # evaluate for winner after player moves
87     #
88
89     $winner = evaluate_board($squares);
90     if ($winner eq "x") {
91         print ("<font color='blue'>Player Won!</font><p>\n");
92         print_table($squares, $page, $winner, $round, $rounds, "final");
93     }else {
94
95     #
96     # check to see if player won. if player won, don't do the rest of this
97
98
99     #
100    # get available choices for computer choices
101    #
102
103        @available_choices = get_available_choices($squares);
104
105    #
106    # get computer move
107    #
108        $computer_move = get_computer_choice(@available_choices);
109        ($x, $y) = split(//, $computer_move);
110        #get coordinates for computer move and store in $x and $y
111        $squares->[$x][$y] = "o";    ## change square to "o"
112        $round_temp = "round" . $round_minus_one;
113        $computer_move_pretty = make_move_pretty($computer_move);
114        $rounds->{$round_temp}->{'computer'} = $computer_move_pretty;
115
116        } #matches else {
117
118    } #matches if ($player_move)
119
120
121    #
122    # now that we have the array with computer and player choices, see if th
ere is a winner
123    #
124    $winner = evaluate_board($squares);
125    if ($winner eq "o") {    #we already checked for x before
126        print ("<font color = 'blue'>Computer Won!</font><p>\n");
127        print_table($squares, $page, $winner, $round, $rounds, "final");
128    }
129

```

```

130     if (($winner ne "o") and ($winner ne "x")) {
131         print ("<font color='blue'>No winner yet</font><p>\n");
132     }
133
134     print_table($squares, $page, $winner, $round, $rounds);
135
136
137     }
138
139     # end table cell
140
141     print ("</td>\n");
142     print ("<td align=middle>\n");
143
144     # get printable versions of moves and print choices
145
146     if (($player_move) or ($computer_move)) {
147         foreach $x(1..$round_minus_one) {
148             $round_temp = "round" . $x;
149             print ("<p><b>Round $x:</b><br>\n");
150             print ("\tplayer: " . $rounds->{$round_temp}->{'player'} . "<br>\n
");
151             print ("\tcomputer: " . $rounds->{$round_temp}->{'computer'} . "<br>
r>\n");
152         }
153     }else {
154         print ("No moves yet.\n");
155     }
156
157     # end table
158
159     print ("</td></tr></table>\n");
160
161     print $page->end_html();
162
163
164     sub player_moves {
165         my $move = $_[0]; ##get move
166         my $page = $_[1]; ## import page object
167         my $squares = $_[2]; ## import array so we can change square
168         my ($x, $y, $z);
169
170         foreach $x(0..2) {
171             foreach $y(0..2) { ##test for each square
172                 if ($move eq "$x$y") {
173                     $squares->[$x][$y] = "x"; ## define array element for choi
ce
174                 }
175             }
176         }
177     }
178
179     sub get_available_choices {
180         my $squares = $_[0];
181         my ($x, $y, $z);
182         my @available_choices = ();
183
184         foreach $x(0..2) {
185             foreach $y(0..2) {
186                 unless (($squares->[$x][$y] eq "x") or ($squares->[$x][$y] eq "
o")) {
187                     $z = "$x$y";
188                     push (@available_choices, $z);
189                 }
190             }
191         }

```

```

192     return @available_choices;
193 }
194
195 sub get_computer_choice {
196     my @available_choices = @_;
197     my $length = @available_choices;
198     my $number = int(rand() * ($length - 1));
199     my $choice = $available_choices[$number];
200     return $choice; ## this will be a coordinate, in the form of $x$y fr
om $z above
201 }
202
203
204 sub print_hidden_values {
205     my $page = $_[0];
206     my $squares = $_[1];
207     my $rounds = $_[2];
208     my ($x, $y, $cell, $round);
209
210     #print hidden values for cells
211     foreach $x(0..2) {
212         foreach $y(0..2) {
213             $cell = "$x$y";
214             print("<input type='hidden' name='$cell' value='" . $squares->
[$x][$y] . "'>");
215             print("\n");
216         }
217     }
218
219     #print hidden values for rounds (history)
220     foreach $x(1..5) {
221         $round = "round" . $x;
222         print("<input type='hidden' name='$round' . \"_x' value ='" . $rou
nds->{$round}->{'player'} . "'>\n");
223         print("<input type='hidden' name='$round' . \"_o' value ='" . $rou
nds->{$round}->{'computer'} . "'>\n");
224     }
225 }
226
227
228 sub evaluate_board {
229     my ($board) = $_[0];
230
231     my @table = (
232         [ 0,0 , 0,1 , 0,2 ],
233         [ 1,0 , 1,1 , 1,2 ],
234         [ 2,0 , 2,1 , 2,2 ],
235         [ 0,0 , 1,0 , 2,0 ],
236         [ 0,1 , 1,1 , 2,1 ],
237         [ 0,2 , 1,2 , 2,2 ],
238         [ 0,0 , 1,1 , 2,2 ],
239         [ 0,2 , 1,1 , 2,0 ],
240     );
241
242     for my $win (@table) {
243         my ($x1, $y1, $x2, $y2, $x3, $y3) = @$win;
244         if ($board->[$x1][$y1] eq $board->[$x2][$y2]
245             && $board->[$x1][$y1] eq $board->[$x3][$y3]) {
246             return $board->[$x1][$y1];
247         }
248     }
249     return;
250 }
251
252 sub print_table {
253     my ($squares, $page, $winner, $round, $rounds, $final) = @_;

```

```

254     my ($visitor, $visitor_name, $time);
255
256     ## print ending table
257
258     print $page->startform(-method=>'POST');
259
260     print_hidden_values($page,$squares,$rounds);
261
262     print("<input type='hidden' name='round', value='$round'>\n");
263
264     print("<table border=1 cellpadding=10>\n");
265     print("<tr valign=middle>\n");
266     for my $row (0..2) {
267         for my $col (0..2) {
268             my $cell = $squares->[$row][$col];
269             if ($cell eq "") {
270                 $cell = $final ? "?"
271                    : "<td><input type='checkbox' name='choice' value='$row$
col'></td>\n";
272             }
273             print("<td align=center>" . $cell . "</td>\n");
274         }
275         print("</tr><tr valign=middle>\n") unless $row == 2;
276     }
277     print("</tr></table><p>\n");
278
279     if ($final) {
280         print("<a href=\"http://www.example.com/cgi-bin/tic_tac.cgi\">Play
Again!</a>");
281     } else {
282         print $page->submit();
283
284         print("<p><font color='red'>Note: if you pick more than one square
, your choice will be the upper and leftmost square that you choose!!</font><p>\n");
;
285     }
286
287     print $page->endform();
288
289
290     print $page->end_html();
291
292     if ($final) {
293         #print log of play
294
295         $visitor = $page->remote_host();
296         if ($visitor =~ /\d*\.\d*\.\d*\.\d*/) {
297             $visitor_name = gethostbyaddr(inet_aton($visitor), AF_INET);
298         }
299
300         $time = localtime(time());
301
302         # open (MAIL, "| /usr/sbin/sendmail -t");
303         # print MAIL "To: author@example.com\n";
304         # print MAIL "Subject: tic tac toe results\n";
305         # print MAIL "\n$visitor, $visitor_name: $time: $winner on round
$round";
306         # close MAIL;
307     }
308
309 }
310
311 sub make_move_pretty {
312     my %squares_names = ("00" => "top left",
313                          "01" => "top center",
314                          "02" => "top right",

```

```
315             "10" => "center left",
316             "11" => "center",
317             "12" => "center right",
318             "20" => "lower left",
319             "21" => "lower center",
320             "22" => "lower right"
321         );
322
323     return $squares_names{$_[0]};
324 }
325
326
```

```
#!/usr/bin/perl
#

#
#A very basic tic-tac-toe program (the computer chooses randomly)
#

use strict;
use CGI;
use Socket 'AF_INET';

sub do_with_board (&);

my ($rounds, $squares, $round, $player_move, $computer_move, $winner);

my $page = CGI->new();

print $page->header;
print $page->start_html();

# print table beginnings

print ("<table width=\"90\%\" border=0 cellpadding=15>\n");
print ("<tr valign=middle>\n");

# left cell is tic tac toe table

print ("<td align=center>\n");

#
# find out which round it is so we know how to define $squares
#

unless ($page->param('round')) {
    $round = 0;
} else {
    $round = $page->param('round');
}

#
# set array of tic tac toe squares
#

do_with_board { $squares->[$a][$b] = $page->param("$a$b") };

#
# set array for determing history of moves (recorded by round)
#

if ($round > 0) {
    for my $rn (1..5) {
        $rounds->{"round$rn"} =
            { player => $page->param("round${rn}_x"),
              computer => $page->param("round${rn}_o"),
            }
    }
}

#
# increment $round to give it a new hidden value
#

my $round_minus_one = $round;
$round = $round + 1;
print ("Round is: $round<br>\n");
```

```
##
## get player move using subroutine (subroutine stores it to $squares array
)
##

$player_move = $page->param('choice');
if ($player_move) {
    my $round_temp = "round" . $round_minus_one;
    player_moves($player_move, $page, $squares);
    my $player_move_pretty = make_move_pretty($player_move);
    $rounds->{$round_temp}->{'player'} = $player_move_pretty;

#
# evaluate for winner after player moves
#

    $winner = evaluate_board($squares);
    if ($winner eq "x") {
        print ("<font color='blue'>Player Won!</font><p>\n");
        print_table($squares, $page, $winner, $round, $rounds, "final");
    }else {

#
# check to see if player won. if player won, don't do the rest of this

#
# get available choices for computer choices
#

        my @available_choices = get_available_choices($squares);

#
# get computer move
#

        $computer_move = $available_choices[int rand @available_choices];
        my ($x, $y) = split(//, $computer_move);
        #get coordinates for computer move and store in $x and $y
        $squares->[$x][$y] = "o"; ## change square to "o"
        $round_temp = "round" . $round_minus_one;
        my $computer_move_pretty = make_move_pretty($computer_move);
        $rounds->{$round_temp}->{'computer'} = $computer_move_pretty;

    } #matches else {
} #matches if ($player_move)

#
# now that we have the array with computer and player choices, see if there
is a winner
#
$winner = evaluate_board($squares);
if ($winner eq "o") { #we already checked for x before
    print ("<font color = 'blue'>Computer Won!</font><p>\n");
    print_table($squares, $page, $winner, $round, $rounds, "final");
}

if (($winner ne "o") and ($winner ne "x")) {
    print ("<font color='blue'>No winner yet</font><p>\n");

    print_table($squares, $page, $winner, $round, $rounds);
}
```

```

}

# end table cell

print ("</td>\n");
print ("<td align=middle>\n");

# get printable versions of moves and print choices

if ($player_move or $computer_move) {
    foreach my $x (1..$round_minus_one) {
        my $round_temp = "round" . $x;
        print ("<p><b>Round $x:</b><br>\n");
        print ("\tplayer: " . $rounds->{$round_temp}->{'player'} . "<br>\n");
        print ("\tcomputer: " . $rounds->{$round_temp}->{'computer'} . "<br>\n");
    }
} else {
    print ("No moves yet.\n");
}

# end table

print ("</td></tr></table>\n");

print $page->end_html();

sub player_moves {
    my ($move, $page, $squares) = @_;

    ## define array element for choice
    do_with_board {$squares->[$a][$b] = "x"};
}

sub get_available_choices {
    my $squares = $_[0];
    my @available_choices = ();

    do_with_board {
        unless (($squares->[$a][$b] eq "x") or ($squares->[$a][$b] eq "o")) {
            push @available_choices, "$a$b";
        }
    };
    return @available_choices;
}

sub print_hidden_values {
    my ($page, $squares, $rounds) = @_;

    #print hidden values for cells
    do_with_board {
        print "<input type='hidden' name='$a$b' value='" . $squares->[$a][$b]
        . "'>\n";
    };

    #print hidden values for rounds (history)
    foreach my $x (1..5) {
        my $round = "round$x";
        print ("<input type='hidden' name='$round' . "_x' value ='" . $rounds->
        >{$round}->{'player'} . "'>\n");
        print ("<input type='hidden' name='$round' . "_o' value='" . $rounds->
        >{$round}->{'computer'} . "'>\n");
    }
}

```

```

}

sub evaluate_board {
    my ($board) = $_[0];

    my @table = (
        [ 0,0 , 0,1 , 0,2 ],
        [ 1,0 , 1,1 , 1,2 ],
        [ 2,0 , 2,1 , 2,2 ],
        [ 0,0 , 1,0 , 2,0 ],
        [ 0,1 , 1,1 , 2,1 ],
        [ 0,2 , 1,2 , 2,2 ],
        [ 0,0 , 1,1 , 2,2 ],
        [ 0,2 , 1,1 , 2,0 ],
    );

    for my $win (@table) {
        my ($x1, $y1, $x2, $y2, $x3, $y3) = @$win;
        if ($board->[$x1][$y1] eq $board->[$x2][$y2]
            && $board->[$x1][$y1] eq $board->[$x3][$y3]) {
            return $board->[$x1][$y1];
        }
    }
    return;
}

sub print_table {
    my ($squares, $page, $winner, $round, $rounds, $final) = @_;
    my ($visitor, $visitor_name, $time);

    ## print ending table

    print $page->startform(-method=>'POST');

    print_hidden_values($page,$squares,$rounds);

    print("<input type='hidden' name='round', value='$round'>\n");

    print("<table border=1 cellpadding=10>\n");
    print("<tr valign=middle>\n");
    for my $row (0..2) {
        for my $col (0..2) {
            my $cell = $squares->[$row][$col];
            if ($cell eq "") {
                $cell = $final ? "?"
                : "<td><input type='checkbox' name='choice' value='$row$col'></td>\n";
            }
            print("<td align=center>" . $cell . "</td>\n");
        }
        print("</tr><tr valign=middle>\n") unless $row == 2;
    }
    print("</tr></table><p>\n");

    if ($final) {
        print("<a href='\"http://www.example.com/cgi-bin/tic_tac.cgi'\">Play Again!</a>");
    } else {
        print $page->submit();

        print("<p><font color='red'>Note: if you pick more than one square, your choice will be the upper and leftmost square that you choose!!</font><p>\n");
    }
}

```

```
print $page->endform();

print $page->end_html();

if ($final) {
    #print log of play

    $visitor = $page->remote_host();
    if ($visitor =~ /\d*\.\d*\.\d*\.\d*/) {
        $visitor_name = gethostbyaddr(inet_aton($visitor), AF_INET);
    }

    $time = localtime(time());

    # open (MAIL, "| /usr/sbin/sendmail -t");
    # print MAIL "To: author@example.com\n";
    # print MAIL "Subject: tic tac toe results\n";
    # print MAIL "\n$visitor, $visitor_name: $time: $winner on round $round";
    # close MAIL;
}

}

sub make_move_pretty {
    my %squares_names = ("00" => "top left",
                         "01" => "top center",
                         "02" => "top right",
                         "10" => "center left",
                         "11" => "center",
                         "12" => "center right",
                         "20" => "lower left",
                         "21" => "lower center",
                         "22" => "lower right"
    );

    return %squares_names{$_[0]};
}

sub do_with_board (&) {
    my $code = shift;
    for $a (0..2) {
        for $b (0..2) {
            $code->();
        }
    }
}
```