Chapter 11

TicTacToe Magic

Three simple games are related in a surprising way. And, the programming of the game play is instructive.

The first of the three games is Pick15. Harold Stark, who was then at the University of Michigan, told me about the game in the late 1960s. I suspect that this is the first time you have heard of it.

The game involves two players. You start by listing the single digit numbers from 1 to 9. You then take turns selecting numbers from the list, attempting to acquire three numbers that add up to 15. Each number can be chosen only once. You may eventually acquire more than three numbers, but you must use exactly three of them to total 15. If neither player can achieve the desired total, the game is a draw.

For example, suppose that Green and Blue are playing. They start with the list.

List	:	1 2 3 4 5 6 7 8 9
Green	:	
Blue	:	

Suppose Green has the first move and chooses 8. Then Blue chooses 4 and Green chooses 2. Now Blue should respond by choosing 5 to prevent Green from getting $2 + 5 + 8 = 15$. Here is the situation after the first two rounds.

List	:	1 ~~2~~ 3 ~~4~~ ~~5~~ 6 7 ~~8~~ 9
Green	:	2 8
Blue	:	4 5

October 2, 2011

Now Green chooses 6 to block $4 + 5 + 6 = 15$ from Blue. This is actually an advantageous move for Green because it gives her two different ways to win, 1+6+8 and $2 + 6 + 7$. Blue cannot block both. If Blue chooses 7, then Green chooses 1 to win. If Blue chooses 1, then Green chooses 7 to win. The final position might be

List	:	~~1~~ ~~2~~ 3 ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ 9
Green	:	2 6 7 8
Blue	:	1 4 5

Note that Green also has $7+8 = 15$, but this does not count because there are only two numbers in the sum.

Figure 11.1 shows the starting position for the Pick15 option in our MATLAB `tictactoe` program. When you play against the computer, your moves are shown in green and the responses from the program are shown in blue.

Figure 11.1. *Starting position for Pick15.*

Figure 11.2 shows the position for our example game after two moves from each player. Green now has to choose 6 to block Blue.

Figure 11.2. *Position after two moves from each player.*

Figure 11.3 shows the final position for our example game. Green has won with 2+6+7 = 15.

Figure 11.3. *Final position for Pick15.*

Please take time out from reading this chapter to try this game a few times yourself, playing against a friend on paper or against our MATLAB program. I think you will find that Pick15 is more challenging than it sounds at first.

The second of our three games is familiar worldwide. It is called "TicTacToe" in the United States, "Noughts and Crosses" in Great Britain, and has many other names in many other countries and languages. Our program `tictactoe` uses green and blue instead of X's and O's. The objective, of course, is to get your color on all three squares in a row, column, or diagonal. Figure 11.6 shows typical output.

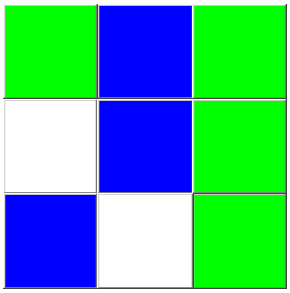

Figure 11.4. *Typical output from a game of TicTacToe. Green has won with the third column.*

Our program `tictactoe` uses a naive three-step strategy.

- If possible, make a winning move.
- If necessary, block a possible winning move by the opponent.
- Otherwise, pick a random empty square.

This strategy will lead to a win only when the opponent makes a error. And, even though it is possible for the player with the second move to always force a draw, this strategy will not accomplish that.

Our third game, Magic15, introduces the Lo-Shu 3-by-3 magic square. Now we see that Pick15 is actually TicTacToe played on a magic square. The rows, columns and main diagonals of the magic square provide all possible ways of having three distinct numbers that sum to 15. Winning moves in Pick15 correspond to winning moves in TicTacToe. All three games are actually the same game with different displays.

8	1	6
3	5	7
4	9	2

Figure 11.5. *Initial configuration for a game of Magic3.*

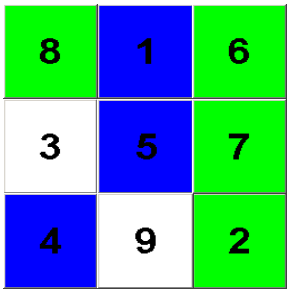

Figure 11.6. *Green has won with 6+7+2 = 15 in the third column.*

Game Play

The `tictactoe` program operates three different views of the same game. You are the green player and the computer is the blue player. The state of the game is carried in a 3-by-3 matrix `X` whose entries are `+1` for cells occupied by green, `-1` for cells occupied by blue, and `0` for as yet unoccupied cells. Obviously, the game begins with `X = zeros(3,3)`

Here is the portion of main program that is executed when you, as the green player, click on one of the buttons or cells. It retrieves `i` and `j`, the matrix indices corresponding to the active button and then checks if the current game already has a winner. If not, it makes your move by setting `X(i,j) = 1` and recursively calls `tictactoe` to let blue take a turn.

```
switch
   case 'green'
      [i,j] = find(gcbo == B);
      if winner(X)
         return
      end
      X(i,j) = 1;
      tictactoe('blue')
```

Here is the corresponding blue portion of the main program. If the current game does not already have a winner, it calls the `strategy` function to get the indices for a move and makes that move.

```
   case 'blue'
      if winner(X)
         return
      end
      [i,j] = strategy(X,-1);
      X(i,j) = -1;
   ...
end
```

Briefly, then, the green moves are determined by user button clicks and the blue moves are determined by the `strategy` function.

Here is the function that checks to see if there is a winner. It looks for any column, row, or diagonal whose elements sum to `3*p` where `p = 1` for green and `p = -1` for blue. The only way a sum can be `3*p` is if all three elements are equal to `p`.

```
function p = winner(X)
% p = winner(X) returns
%    p = 0, no winner yet,
%    p = -1, blue has won,
%    p = 1, green has won,
%    p = 2, game is a draw.

for p = [-1 1]
   s = 3*p;
   win = any(sum(X) == s) || any(sum(X') == s) || ...
         sum(diag(X)) == s  || sum(diag(fliplr(X))) == s;
   if win
      return
   end
end
p = 2*all(X(:) ~= 0);
```

Here is the naive, but fairly effective strategy that the computer uses against you. It first checks to see if it can make a winning move. If not, it then checks to see if it needs to block your pending winning move. It neither of these happens, it simply chooses a random empty cell. This crude strategy makes the play interesting. It is possible for you to beat the computer if its random choices are not optimal. The exercises suggest some improvements in this strategy.

```
function [i,j] = strategy(X,p);
% [i,j] = strategy(X,p) is a better, but not perfect, move for player p.

% Appear to think.
pause(0.5)

% If possible, make a winning move.
[i,j] = winningmove(X,p);

% Block any winning move by opponent.
if isempty(i)
   [i,j] = winningmove(X,-p);
end

% Otherwise, make a random move.
if isempty(i)
```

```
    [i,j] = find(X == 0);
    m = ceil(rand*length(i));
    i = i(m);
    j = j(m);
end
```

Finally, here is the function that seeks a winning move by either player. It looks for rows, columns, or diagonals with sums equal to `2*p` for either value of `p`.

```
function [i,j] = winningmove(X,p);
% [i,j] = winningmove(X,p) finds any winning move for player p.

s = 2*p;
if any(sum(X) == s)
   j = find(sum(X) == s);
   i = find(X(:,j) == 0);
elseif any(sum(X') == s)
   i = find(sum(X') == s);
   j = find(X(i,:) == 0);
elseif sum(diag(X)) == s
   i = find(diag(X) == 0);
   j = i;
elseif sum(diag(fliplr(X))) == s
   i = find(diag(fliplr(X)) == 0);
   j = 4 - i;
else
   i = [];
   j = [];
end
```

The remainder of the `tictactoe` function is responsible for the gui.

Let's see how all this works on an example. We begin with the position shown in figure 11.2, after two moves by each player. The state of the game at this point is

```
X =
     1     0     0
     0    -1     0
    -1     0     1
```

It is green's move. Green needs to click the `6` button to prevent blue from having `4+5+6 = 15`. Or, it is probably easier to look at `M = magic(3)`

```
M =
     8     1     6
     3     5     7
     4     9     2
```

You see that 4 and 5 are on the antidiagonal, so green needs the 6 to prevent a win in the TicTacToe version of this game. Either way, this corresponds to `(i,j) = (1,3)` and setting `X(i,j) = 1`.

```
X =
     1     0     1
     0    -1     0
    -1     0     1
```

Now it is blue's turn. Both the first row and the third column of `X` have sums equal to 2, corresponding to the fact that green has two pending winning moves. With `p = -1`, the `strategy` function looks for a block by calling `winningmove(X,-p)`. The result is `(i,j) = (2,3)`, although it just as well could have been `(i,j) = (1,2)` if `winningmove` made its checks in another order. This leads to

```
X =
     1     0     1
     0    -1    -1
    -1     0     1
```

On the next turn, green finds that the sum along the first row is 2, and so sets `X(1,2) = 1` to make the sum 3. This gives

```
X =
     1     1     1
     0    -1    -1
    -1     0     1
```

and green proclaims a win.

It would be possible to implement recursive backtracking strategies like we describe in the Sudoku chapter. But there are only a few hundred possible games, so the backtracking would be exhaustive. And boring – all the games would be draws.

Recap

```
%% TicTacToe Chapter Recap
% This is an executable program that illustrates the statements
% introduced in the TicTacToe Chapter of "Experiments in MATLAB".
% You can access it with
%
%    tictactoe_recap
%    edit tictactoe_recap
%    publish tictactoe_recap
%
% Related EXM programs
%
%    tictactoe
```

```
%% tictactoe/winner

%  function p = winner(X)
%  % p = winner(X) returns
%  %    p = 0, no winner yet,
%  %    p = -1, blue has won,
%  %    p = 1, green has won,
%  %    p = 2, game is a draw.
%
%  for p = [-1 1]
%     s = 3*p;
%     win = any(sum(X) == s) || any(sum(X') == s) || ...
%           sum(diag(X)) == s  || sum(diag(fliplr(X))) == s;
%     if win
%        return
%     end
%  end
%  p = 2*all(X(:) ~= 0);

%% tictactoe/strategy

%  function [i,j] = strategy(X,p);
%  % [i,j] = strategy(X,p) is a move for player p.
%
%  % Appear to think.
%  pause(0.5)
%
%  % If possible, make a winning move.
%  [i,j] = winningmove(X,p);
%
%  % Block any winning move by opponent.
%  if isempty(i)
%     [i,j] = winningmove(X,-p);
%  end
%
%  % Otherwise, make a random move.
%  if isempty(i)
%     [i,j] = find(X == 0);
%     m = ceil(rand*length(i));
%     i = i(m);
%     j = j(m);
%  end

%% tictactoe/winningmove
```

```
%  function [i,j] = winningmove(X,p);
%  % [i,j] = winningmove(X,p) finds any winning move for player p.
%
%  s = 2*p;
%  if any(sum(X) == s)
%     j = find(sum(X) == s);
%     i = find(X(:,j) == 0);
%  elseif any(sum(X') == s)
%     i = find(sum(X') == s);
%     j = find(X(i,:) == 0);
%  elseif sum(diag(X)) == s
%     i = find(diag(X) == 0);
%     j = i;
%  elseif sum(diag(fliplr(X))) == s
%     i = find(diag(fliplr(X)) == 0);
%     j = 4 - i;
%  else
%     i = [];
%     j = [];
%  end
```

Exercises

11.1 *Traditional.* Modify `tictactoe.m` so that it uses traditional X's and O's.

11.2 *Win.* Is it possible to win against `tictactoe` with its naive strategy?

11.3 *First move.* Modify `tictactoe` so that the computer takes the first move.

11.4 *Center square.* Modify the strategy used by `tictactoe.m` so that, before taking a random move, it takes the center square if it is available. Does this improve the program's chances of winning or forcing a draw?

11.5 *xkcd.* Implement the complete tictactoc strategy available from the Web comic strip `xkcd` by Randal Munroe at

`http://xkcd.com/832`

11.6 *Computer versus computer.* Modify `tictactoe.m` so that the computer plays against itself. Count the number of draws and wins for both sides. Run a large number of games, with and without the addition of the center square strategy.