27th Annual Rowan University Programming Contest

hosted by the
Computer Science Department

Friday, 26 April 2013

Contest Problem

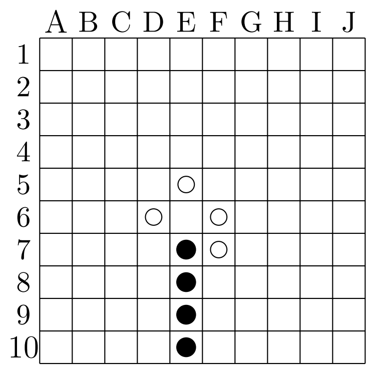
Rowan
University™

1 Introduction

The game *Battleship* involves two players with ships arranged on a grid, taking turns attempting to sink each other's fleets. As most commonly played, each player has five ships: an *Aircraft Carrier* of length 5 (abbreviated *A*), a *Battleship* of length 4 (*B*), a *Destroyer* (*D*) and a *Submarine* (*S*), both length 3, and a *PT Boat* (*P*) of length 2. Each ship is placed horizontally or vertically on a 10×10 grid, without going off the side. A turn consists of naming a square, which the opponent declares to be a hit or a miss. When a ship has been hit along its full length, it declared to have been sunk, and it is identified.

This contest considers the logic of choosing the best shot. There are two situations to be considered: when you are trying to find a ship, and when you have hit a ship but not yet sunk it.

1.1 No Damaged-But-Unsunk Ships



Consider the game in progress at right. The opponent's Battleship has been sunk, and the other four ships are being sought.

We need to find the square most likely to have a ship on it. Those which already have shots (either hit or miss) get a value of zero, as does E6, because no ship can be placed in a way where it will cover E6.

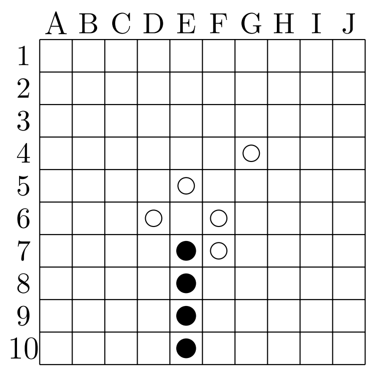
Consider D7. There are two possible positions for *P* which include D7: C7-D7 and D7-D8 (it may have one end on the square and is either horizontal or vertical). Using the same reasoning, there are two possible positions for both *D* and *S* which include D7: B7-D7 and D7-D9. Possible placements for *B* are zero. There is no location for *A* such that it covers D7, because it's too long. That gives 6 ship placements which include D7 (two each for the three vessels which fit).

Moving down and to the left, there are 17 possible placements that include C8. *P* could be at any of: B8-C8, C8-D8, C7-C8, C8-C9. For *D* and *S*, possibilities include A8-C8, B8-D8, C6-C8, C7-C9, C8-C10. For *A*, there are no horizontal possibilities, but it could be at any of: C4-C8, C5-C9, C6-C10. (Again, *B* is not considered.)

Moving up that column, C4 is included in 4 placements for *P* (C3-C4, C4-C5, B4-C4, C4-D4), 6 each for *D* and *S*, (C2-C4, C3-C5, C4-C6, A4-C4, B4-D4, C4-E4), and 7 for *A* (C1-C5, C2-C6, C3-C7, C4-C8, A4-E4, B4-F4, C4-G4), giving a total of 23.

Now consider G4: There are 4 placements for *P* which include that square; *D* and *S* each have 6 placements which include G4; *A* has 8 placements covering G4. G4 is therefore included in 24 possible ship placements.

In this situation, your program should print that the best move is G4, with a value of 24.



Now let us assume that the player has chosen G4, and that it was a miss. That would give us the board position at right.

The number of theoretical ship placements which includes G4 is now zero.

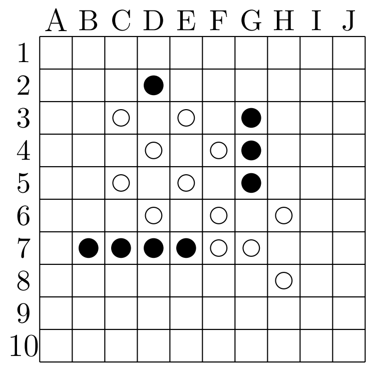
The numbers for D7 and C8 remain as they were, 6 and 17, respectively.

The number for C4 has gone down by one: *P*, *D*, and *S* are as before, but *A* can no longer be placed at C4-G4, because G4 is no longer available. C4's value is now 22.

Consider C3; it is included in 4 placements for P, 6 for each of S and D, and 6 for A. Its value is also 22. The same is true of F3, H3, and H5.

In a case like this, any of C3, F3, H3, C4, and H5 is equally good. Presented with this board layout, your program should print out all five positions and their values, in the order C3, F3, H3, C4, and H5.

1.2 Damaged-But-Unsunk Ships



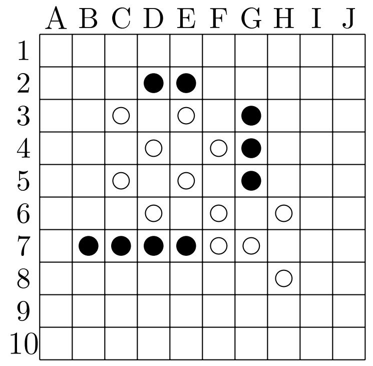
Consider the board at right. The opponent's Battleship and Submarine have been sunk, and there's a hit at D2. It must be one of either P, D, or A. (As B and S are sunk, they are not possibilities.) The best choice now is a square around D2.

Consider D1 and D3. P can fit vertically and touch either square, and D can fit vertically and touch both. A cannot fit vertically in the space. So there are two ship placements which include D1 (one each for P and D), and two which include D3.

Next consider C2. P can be on C2. D can be on C2 in two different ways: it might cover B2-D2, or C2-E2. A can be on C2 in three ways: A2-E2, B2-F2, C2-G2. So for square C2, the total number of ship placements which include that square is 6. That makes C2, with 6 possible placements, better than either D1 or D3, which each have only two.

But then consider E2. P can be on E2. D can be on E2 in two different ways: it might cover C2-E2 or D2-F2. And A can be on E2 in 4 ways: A2-E2, B2-F2, C2-G2, D2-H2. The total number of possible ship placements which include E2 is 7, making it the best choice.

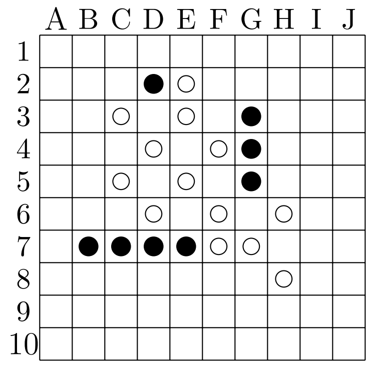
In this situation, your program should report that the best move is E2 with a score of 7.



Suppose the player has moved to E2 and it is a hit, but the ship was not sunk. We know that the ship must be either A or D, because if it had been P it would have been sunk. It is possible, but unlikely, that there are two different ships right next to each other. We will assume that is not the case in a situation like this, but be sure to see the next section where it might come up.

That leaves us with either C2 or F2. D can be on C2 in one way (C2-E2), and it can be on F2 in one way (D2-F2). There are three ways A can be on C2, and three ways it can be on F2.

In this case, your program should report that both C2 and F2 have a score of 4.



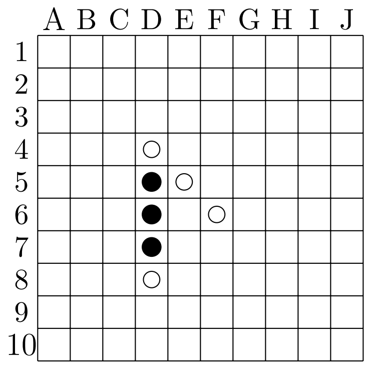
Alternatively, suppose E2 had been a miss? We know that the ship must be either D or P, because A won't fit in the spaces which are left.

D1 and D3 have the same values as before: there are two ship placements which include those squares, one each for D and P.

The value for C2 has dropped; there are no placements for A which include C2; there's still one for P, and there's also only one for D. (Those which included E2 are no longer valid.)

With this board position, your program should print that D1, C2, and D3 have a value of 2.

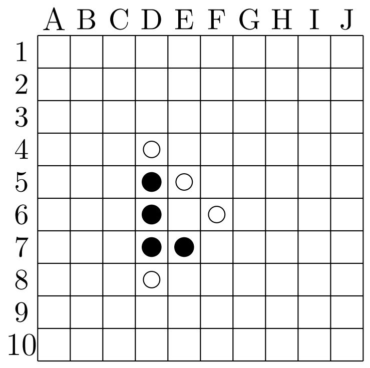
1.3 Multiple Damaged-But-Unsunk Ships



If multiple ships are next to each other, it's possible to shoot across them, and thus cause damage to multiple vessels without sinking any of them.

In the position at right, there are no sunk ships. The player got a hit on their third move at D6. Since there were more ways for a ship to be a vertical than horizontal, the player chose D5 and then D4, which was a miss. Then the player chose D7, a hit, and D8, a miss.

In this situation, each of the hits has to be considered independently: there are four ship placements which include C5 (each ship except *A* going horizontally from D5); there are 8 placements which include C6; there are 5 which include E6; there are 11 which include C7; there are 12 which include E7. So in this position, the best choice is E7.



Suppose E7 is a hit, but no ship is sunk as a result.

There is the possibility that D7-E7 are any of the ships but *P* (because it would be sunk), and also the possibility that one ship goes left from D7 and another ship goes right (or down) from E7.

You should assume that any string of hits on unsunk ships are a single ship, unless there is a miss at both ends. Further, the longest string of hits should always be pursued first, because sinking a ship gives information which can be used in calculations about other hits.

In this situation, we have three strings of hits on three different ships; as D7-E7 is the longest, the only shots your program should consider are C7 and F7. For C7, there are three placements for *A*, two for *B*, one each for *D* and *S*, and none for *P* (remember we are assuming that D7 and E7 are hits on a single ship). The same applies to F7. So your program should report that the best choice is a tie between C7 and F7, each with a value of 7.

Your challenge for this contest is to write a program which reads in the information about ships still in play and the moves already made on the board, and prints out the position(s) remaining which are included in the greatest number of possible placements for all the remaining ships.

Notes:

A damaged-but-not-sunk ship always takes highest priority when choosing the next move.

If there are several damaged-but-not-sunk ships, and one has a longer series of hits than any others, it takes priority.

If there are several damaged-but-not-sunk ships, and all have the same amount of damage, then the spaces around each need to be considered.

If there are multiple squares with the same value, they should be listed in order by row, lowest row first. If several are on the same row, they should be listed in order by column.

2 Input

2.1 Input Specification

For text input, your program should accept input in the following format:

1. An integer, $\mathcal{D}$, where $1 \le \mathcal{D} \le 50$, which is the number of datasets in this file.
2. $\mathcal{D}$ data sets, each of which is in this format:
   (a) One line with one integer, $\mathcal{S}$, where $1 \le \mathcal{S} \le 5$, which is the number of ships still in play.
   (b) One line with $\mathcal{S}$ letters, from the set {A, B, D, S, P}, indicating which ships are in play. If there is more than one letter, they will be separated by a single space. There will always be at least one letter.
   (c) 10 lines, each with 10 integers, one of {0, 1, 2, 3}, laying out which squares have been used. A '0' is an empty space, a '1' is miss, a '2' is a hit, and a '3' represents a sunken ship.

2.2 Sample Input #1

Data in file	Item #	Meaning in plain English
`2`	1	*this file has 2 data sets*
`4`	2a	*Data Set 1 has 4 ships still in play*
`A D S P`	2b	*the ships being sought*
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 1 0 0 0 0 0`	2c	*the board's current status*
`0 0 0 1 0 1 0 0 0 0`		
`0 0 0 0 3 1 0 0 0 0`		
`0 0 0 0 3 0 0 0 0 0`		
`0 0 0 0 3 0 0 0 0 0`		
`0 0 0 0 3 0 0 0 0 0`		
`4`	2a	*Data Set 2 has 4 ships still in play*
`A D S P`	2b	*the ships being sought*
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 1 0 0 0`		
`0 0 0 0 1 0 0 0 0 0`	2c	*the board's current status*
`0 0 0 1 0 1 0 0 0 0`		
`0 0 0 0 3 1 0 0 0 0`		
`0 0 0 0 3 0 0 0 0 0`		
`0 0 0 0 3 0 0 0 0 0`		
`0 0 0 0 3 0 0 0 0 0`		

(This input, which corresponds to the diagrams in §1.1, is on the website as **sample1.txt**.)

2.3 Sample Input #2

Data in file	Item #	Meaning in plain English
`3`	1	*this file has 3 data sets*
`3`	2a	*Data Set 1 has 3 ships still in play*
`A D P`	2b	*the ships being sought*
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 2 0 0 0 0 0 0`		
`0 0 1 0 1 0 3 0 0 0`		
`0 0 0 1 0 1 3 0 0 0`		
`0 0 1 0 1 0 3 0 0 0`	2c	*the board's current status*
`0 0 0 1 0 1 0 1 0 0`		
`0 3 3 3 3 1 1 0 0 0`		
`0 0 0 0 0 0 0 1 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`3`	2a	*Data Set 2 has 3 ships still in play*
`A D P`	2b	*the ships being sought*
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 2 2 0 0 0 0 0`		
`0 0 1 0 1 0 3 0 0 0`		
`0 0 0 1 0 1 3 0 0 0`		
`0 0 1 0 1 0 3 0 0 0`	2c	*the board's current status*
`0 0 0 1 0 1 0 1 0 0`		
`0 3 3 3 3 1 1 0 0 0`		
`0 0 0 0 0 0 0 1 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`3`	2a	*Data Set 3 has 3 ships still in play*
`A D P`	2b	*the ships being sought*
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 2 1 0 0 0 0 0`		
`0 0 1 0 1 0 3 0 0 0`		
`0 0 0 1 0 1 3 0 0 0`		
`0 0 1 0 1 0 3 0 0 0`	2c	*the board's current status*
`0 0 0 1 0 1 0 1 0 0`		
`0 3 3 3 3 1 1 0 0 0`		
`0 0 0 0 0 0 0 1 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		

(This input, which corresponds to the diagrams in §1.2, is on the website as **sample2.txt**.)

2.4 Sample Input #3

Data in file	Item #	Meaning in plain English
`2`	1	*this file has 2 data sets*
`5`	2a	*Data Set 1 has 5 ships still in play*
`A B D S P`	2b	*the ships being sought*
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 1 0 0 0 0 0 0`		
`0 0 0 2 1 0 0 0 0 0`	2c	*the board's current status*
`0 0 0 2 0 1 0 0 0 0`		
`0 0 0 2 0 0 0 0 0 0`		
`0 0 0 1 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`5`	2a	*Data Set 2 has 5 ships still in play*
`A B D S P`	2b	*the ships being sought*
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 1 0 0 0 0 0 0`		
`0 0 0 2 1 0 0 0 0 0`	2c	*the board's current status*
`0 0 0 2 0 1 0 0 0 0`		
`0 0 0 2 2 0 0 0 0 0`		
`0 0 0 1 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		
`0 0 0 0 0 0 0 0 0 0`		

(This input, which corresponds to the diagrams in §1.3, is on the website as **sample3.txt**.)

Notes:
The game board will always be 10×10, the ships will be represented by the letters 'A', 'B', 'D', 'S', and 'P', and their lengths will be, respectively, 5, 4, 3, 3, and 2.

There will be no case in which there are separate groups of hits in a grid. Any '2', representing a ship hit-but-not-sunk, will either be the only one in on the grid, or it will be next to another '2'.

You may choose to have your program read the input from the keyboard, or ask the user for a filename and then read the file. Users of GUI-based programming environments may prefer to use text boxes into which the values can be entered, and buttons to begin their calculation. Any reasonable variation in the spirit of the problem is acceptable.

You need not do error-checking on the input. Each line will have exactly the number of items described with no stray characters. There will be no blank lines.

3 Output

3.1 Output Specification

For each data set configuration, your program must generate output as follows:

1. The text '`Analyzing` *D* `data sets`', where *D* is the number of data sets in the input.
2. For each data set:
   (a) The text '`Data Set` *S*', where *S* is the number of the data set being reported on.
   (b) The text '`Best Move Value:` *V* `at` *LN*', where *L* is the letter of the column for the best move and *N* is the number of the row.
   If there is more than one square with the same score, list them separated by commas, in order first by row and then by column.

3.2 Sample Output 1

```
Analyzing 2 data set(s)
Data Set 1
   Best Move Value: 24 at G4
Data Set 2
   Best Move Value: 22 at C3, F3, H3, C4, H5
```

(This output corresponds to Sample Input 1 from page 5.)

3.3 Sample Output 2

```
Analyzing 3 data set(s)
Data Set 1
   Best Move Value:  7 at E2
Data Set 2
   Best Move Value:  4 at C2, F2
Data Set 3
   Best Move Value:  2 at D1, C2, D3
```

(This output corresponds to Sample Input 2 from page 6.)

3.4 Sample Output 3

```
Analyzing 2 data set(s)
Data Set 1
   Best Move Value: 12 at E7
Data Set 2
   Best Move Value:  7 at C7, F7
```

(This output corresponds to Sample Input 3 from page 7.)

Your output does **not** have to duplicate the sample output as regards spacing or use of upper/lower case. Your output should be neat, but need not exactly match the sample.

4 Test Data

Run your program on this input and print the results. **You must submit printed output to earn full points.** Your program will also be run on data known only to the judges.

4.1 Test Input #1

```
4
5
A B D S P
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
5
A B D S P
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
5
A B D S P
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 2 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
2
A B
1 0 1 0 1 0 0 0 0 1
0 0 1 0 0 1 0 0 0 0
0 1 3 0 0 0 1 0 0 0
1 0 3 1 0 0 0 1 0 0
1 0 3 0 1 1 0 0 1 0
0 1 0 0 1 3 3 0 0 1
0 0 1 0 0 1 1 0 1 0
1 0 0 1 0 0 0 1 3 0
0 0 0 0 1 0 0 0 3 0
0 0 0 0 0 1 0 0 3 1
```

4.2 Test Input #2

```
4
1
P
3 3 3 0 1 0 1 0 1 0
0 1 0 1 0 1 0 1 0 1
1 0 1 0 1 0 1 0 1 0
0 1 0 1 1 0 0 1 0 1
1 0 1 0 3 0 1 0 1 0
0 1 0 0 3 1 1 0 0 0
0 0 1 0 3 0 3 3 3 3
0 1 1 0 3 0 1 0 0 0
3 3 3 0 3 0 0 0 1 0
0 0 0 0 0 0 0 1 0 1
5
A B D S P
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
5
A B D S P
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
0 0 0 0 2 1 0 0 0 0
0 0 0 0 2 0 1 0 0 0
0 0 0 0 2 0 0 1 0 0
0 0 0 0 2 0 0 0 0 0
0 0 0 0 2 0 0 0 0 0
0 0 0 0 1 0 0 0 0 0
2
A B
1 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 1 3 0 0 0 0 0 0 0
1 0 3 1 0 0 0 0 0 0
0 0 3 0 1 1 0 0 0 0
0 0 0 0 1 3 3 0 0 0
0 0 0 0 0 1 1 0 1 0
0 0 0 0 0 0 0 1 3 0
0 0 0 0 0 0 0 0 3 0
0 0 0 0 0 0 0 0 3 1
```

All sample and test data sets are available at <http://elvis.rowan.edu/rupc/2013>