

Test sheet 5 (battleship)

by Connor Johnson

Submission date: 09-Jan-2026 02:46PM (UTC+0000)

Submission ID: 272687674

File name: 65019_Connor_Johnson_Test_sheet_5__battleship__703116_149978028.docx (106.18K)

Word count: 764

Character count: 4152

Name: Connor Johnson

Student Number: 2517511

WORKSHEET 5 – TESTING (Battleships)

For this worksheet, the program can be implemented in a number of different ways and demonstrate different features.

Create your own test set to demonstrate strengths and/or limitations of your solution. Use as many of the template boxes below as you need.

Test:

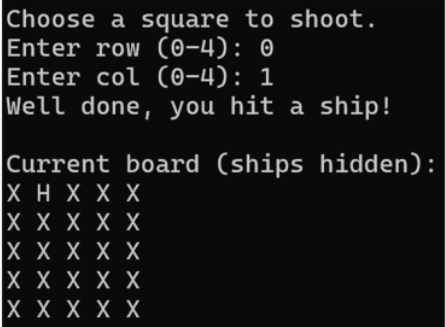
Game behaviour (Miss)

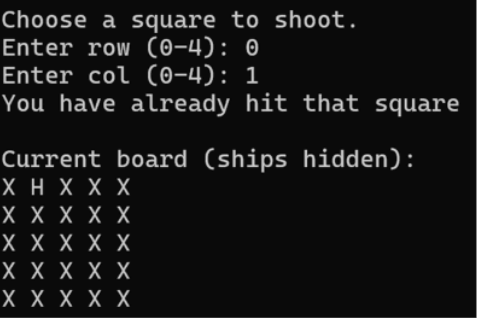
Screenshot(s) of output:

```
Choose a square to shoot.  
Enter row (0-4): 0  
Enter col (0-4): 4  
Miss, you hit the sea  
  
Current board (ships hidden):  
X X X X X  
X X X X X  
X X X X X  
X X X X X  
X X X X X
```

Result: Did the programme behave as expected ie syntactically and logically correct?

The programme behaved as expected here. The user selected a valid coordinate between 0-4 that corresponds to a grid marker 'X'. The programme correctly identified this as a miss and displayed 'Miss, you hit the sea'. This left the board unchanged with all ships still hidden and the game continued to the next turn without errors. This shows that both the input and game logic for a miss are working correctly.

Test:
Game behaviour (Hit)
Screenshot(s) of output:  <pre> Choose a square to shoot. Enter row (0-4): 0 Enter col (0-4): 1 Well done, you hit a ship! Current board (ships hidden): X H X X X X X X X X X X X X X X X X X X X X X X X </pre>
Result: Did the programme behave as expected ie syntactically and logically correct? The programme behaved as expected in this test. The user entered a valid coordinate within the range of 0-4 that corresponded to a ship location that is then marked with 'H'. the programme correctly detected the hit and displayed the message 'Well done, you hit a ship!'. The game then continued normally with the board still hiding remaining ship locations. This confirms that the hit detection and grid updating logic are working correctly.

Test:
Game behaviour (Already Hit)
Screenshot(s) of output:  <pre> Choose a square to shoot. Enter row (0-4): 0 Enter col (0-4): 1 You have already hit that square Current board (ships hidden): X H X X X X X X X X X X X X X X X X X X X X X X X </pre>
Result: Did the programme behave as expected ie syntactically and logically correct? The programme again behaved as expected here, the user selected a coordinate that had already been hit previously and marked with 'H'. the programme correctly detected that the location had been previously hit and displayed the message 'You have already hit that square' showing the programme correctly prevents the same square from being counted as hit multiple times.

Test:
Input validation (out of range)
Screenshot(s) of output: <pre> Choose a square to shoot. Enter row (0-4): 5 Enter col (0-4): 5 Invalid coordinates! Must be between 0 and 4. Current board (ships hidden): X H X X X X X X X X X X X X X X X X X X X X X X X </pre>
Result: Did the programme behave as expected ie syntactically and logically correct? In this test the programme behaved as expected. The user entered two coordinates outside of the valid input range of 0-4. The programme correctly detected this as an invalid input and displayed an appropriate error message 'Invalid coordinates! Must be between 0 and 4.'. The game did not crash and continued to ask for another input within appropriate range. This confirms that the input validation for numbers outside of the allowed ranges are working correctly.

Test:
Game win
Screenshot(s) of output: <pre> Choose a square to shoot. Enter row (0-4): 4 Enter col (0-4): 3 Well done, you hit a ship! Current board (ships hidden): X H X X X X H X H H X H X X X X X X X X X H H H X </pre> All ships have been hit. Game over!
Result: Did the programme behave as expected ie syntactically and logically correct? During this test, the programme behaved as expected. The remaining ship was 'destroyed' leaving no squares for continued attempts. As a result, the game loop successfully ended, and the programme displayed the correct message 'All ships have been hit. Game over!'. The final board state was displayed with all ship positions marked as 'H' confirming the user has won. This shows that the win condition logic inside the programme can correctly detect when all ships have been destroyed.

Final code:

```
#pragma once
#include <stdio.h>
#define N 5

void printHiddenGrid(char grid[N][N])
{
    printf("\nCurrent board (ships hidden):\n");
    for (int r = 0; r < N; r++)
    {
        for (int c = 0; c < N; c++)
        {
            if (grid[r][c] == 'S')
                printf("X ");
            else
                printf("%c ", grid[r][c]);
        }
        printf("\n");
    }
}

int countShipsLeft(char grid[N][N])
{
    int count = 0;
    for (int r = 0; r < N; r++)
    {
        for (int c = 0; c < N; c++)
        {
            if (grid[r][c] == 'S') count++;
        }
    }
    return count;
}
```

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "battleshipFunc.h"

void main(void)
{
    char grid[N][N] =
    {
        {'X','S','X','X','X'},
        {'X','S','X','S','S'},
        {'X','S','X','X','X'},
        {'X','X','X','X','X'},
        {'X','S','S','S','X'}
    };

    while (countShipsLeft(grid) > 0)
    {
        printHiddenGrid(grid);

        int row, col;
        printf("\nChoose a square to shoot.\n");
        printf("Enter row (0-4): ");
        if (scanf("%d", &row) != 1)
        {
            printf("Invalid input. Please enter a number.\n");
            while (getchar() != '\n');
            continue;
        }

        printf("Enter col (0-4): ");
        if (scanf("%d", &col) != 1)
        {
            printf("Invalid input. Please enter a number.\n");
            while (getchar() != '\n');
            continue;
        }

        if (row < 0 || row >= N || col < 0 || col >= N)
        {
            printf("Invalid coordinates! Must be between 0 and 4.\n");
            continue;
        }
    }
}

```

```

        if (grid[row][col] == 'X')
        {
            grid[row][col] = 'M';
            printf("Miss, you hit the sea\n");
        }
        else if (grid[row][col] == 'S')
        {
            grid[row][col] = 'H';
            printf("Well done, you hit a ship!\n");
        }
        else if (grid[row][col] == 'H')
        {
            printf("You have already hit that square\n");
        }
        else if (grid[row][col] == 'M')
        {
            printf("You have already tried that square\n");
        }
    }

    printHiddenGrid(grid);
    printf("\nAll ships have been hit. Game over! \n");
}

```

Test sheet 5 (battleship)

GRADEMARK REPORT

FINAL GRADE

18/0

GENERAL COMMENTS

WORKSHEET 5

Coding- You have created a reasonable battleships game with ability to sink ships and end the game once all have been sunk.

You are missing more advanced features such as randomising the game (this in turn brings problems to solve, such as boats not overlapping on the grid)

Good use of the function for printing the board.

I'm not sure about countshipsleft() - you could instead just take one off a total since we know how many ships are there, this would be more efficient.

Good to see validation of the input.

9/15

Testing - Could have been a little more extensive, eg show a minus number. However generally good based on the code produced.

9/15

Overall mark Worksheet 5: 18/30

**Organisation of report, writing style,
presentation of work**

Much better writing style in Worksheets 3, 4, 5, good to see you taking on board feedback from Worksheets 1 + 2 (and maybe James's module). Overall layout good.

8 /10

Overall mark for assignment (5 worksheets) -
76%

Test mark - 82%

Overall mark for module: 76 %

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6
