ECEN 1310 Project: Battleship

Aniruddha Phatak

Project office hours: Thursday 6-7 pm in ECEE 1B61A.

1 General Overview

In this project, you will write a C program in which a single player plays a game of Battleship with the computer. If you are not aware of this game, the basic rules are given in the last section. In your project, the computer will start by randomly arranging a fleet of ships on a square board (this arrangement will not be visible the player) and the player will have a certain limited number of shots to hit all the cells occupied by ships on the board. The player wins if he/she hits all the cells occupied by ships and the computer wins if the player runs out of shots before hitting all the cells occupied by ships.

Here are some general rules/guidelines that you should follow:

1. In your game, assume the following fleet and the size of each ship (the size refers to the number of cells that each ships occupies):

   - Aircraft carrier - 4
   - Battleship - 3
   - Cruiser - 3
   - Submarine - 2

   For example, a cruiser occupies 4 consecutive cells. Ships can only be placed horizontally or vertically and may not overlap with each other.

2. Your program should take the size of the board (optional) and the maximum number of shots allowed as command line input (the board size if provided should be a single integer since it is always a square.) For example if your command line input is "6   20", then the computer should randomly arrange the ships on a 6×6 board and the player is allowed a maximum of 20 shots. If only a single integer is provided as input, then that number should be interpreted as the maximum shots allowed and the board size should take the default value of 8.

3. The minimum size of the board should be 5×5 and the maximum size should be 20×20. The minimum value of the number of shots allowed

should obviously be at least equal to 12, size there are 12 cells occupied by ships on the computer's board. Also, this number should be at most equal to $n^2 - 1$, where $n \times n$ is the size of the board (if the player is allowed n^2 or greater number of shots, they will always win since they can simply hit all the cells on the board.) Your program should have all these checks and must print a message indicating the correct usage if the user tries to input a wrong value.

4. The rows of the board may be indexed by numbers (like row 1, row 2 and so on) and the columns may be indexed by letters (like column a, column b and so on.) Hence each cell on the board can be referred to by a combination of a number followed by a letter (like cell "4e".) The player can then enter the cell that he wants to hit by simply entering such a combination (see the example gameplay given.)

5. After each shot, the computer should print whether the shot hit any of the ships or not (for example, by printing "That's a Hit!" or "That's a Miss!".) Once all the cells occupied by a particular ship have been hit, the computer should announce this (by printing, for example, something like "You sunk by Cruiser!".)

6. After each shot, your program should display a matrix-like structure to the user that indicates the cells that were successful hits by a "X", cells that were shot at unsuccessfully by a "O" and the cells that are not yet shot at by a "$\sim$". This way, the player can keep track of which cells are yet to be hit and which cells are likely to contain ships.

7. If after any particular shot, the number of shots left is less than the number of cells occupied by ships, the program should print that player lost and the game ends.

2 Example Gameplay

Here is an example of how your battleship program should look like on the terminal. You don't have to print out these exact words and symbols, but your game should proceed in a similar logical fashion. Suppose that your command like input is "7   25". Then the terminal would look something like:

```
The computer has finished arranging the fleet! Let's play!

~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~

You have 25 shots remaining. Take your next shot!

>>2e

That's a hit! The board now looks like:

~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ X ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~

You have 24 shots remaining. Take your next shot!

>> 1a

That's a miss! The board now looks like:

O ~ ~ ~ ~ ~ ~
~ ~ ~ ~ X ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~

You have 23 shots remaining. Take your next shot!

>>
```

3 Tracking your progress

3.1 Checkpoint (40 % of the points): Fleet arrangement

At the checkpoint, your program should be able to:

1. Parse the command line input with the necessary checks to catch any faulty input and display a usage message for any faulty input.
2. Generating the random arrangement of the fleet of ships. You can use the `rand()` function from the `stdllib` library to generate random numbers.

3.2 Final submission (60 % of the points): Rest of the game

This part involves the rest of the game after the parsing of the input arguments and the random fleet arrangement. Hence your program should:

1. Accept user input for the cell the be hit ("4e") and display the appropriate message of a hit or a miss. If all the cells of a ship have been hit, announce the sinking of that ship.
2. After each shot display the matrix indicating the successful hits (X), unsuccessful shots (O) and cells yet to be shot at (∼).
3. After each shot, check if number of shots left are at least equal to the number of cells with ships and end the game if not.
4. Print the winner (computer wins if all ships are sunk, player wins if the number of shots remaining reaches zero.)

4 Some useful tips:

1. Start by imagining the high-level architecture of your project: the header file ("**battleship.h**" that defines the interfaces of the functions that you will use), source code file ("**battleship.c**" that has the implementation of these functions) and the main file ("**mainbattleship.c**" from which you will call these library functions.)
2. You can define the board as a new data type like you defined the matrix data type as follows:

```
struct _board{
   int size; \\size of the board
   int *data;  \\data indicating the fleet arrangement
   int cells_with_ships;  \\cells left with ships
   int carrier; \\cells with aircraft_carrier
   int battleship; \\cells with battleship
   int cruiser;  \\cells with cruiser
   int submarine;  \\ cells with submarine
   };
```

3. To come up with a list of functions you would need to define, sequentially imagine how the game would be played and the actions that you would need to take in the code, starting from the command-line inputs by the user. Clearly, you need a `parseArgs` function to parse the input arguments from the command line. Once the correct command-line arguments have been parsed, you need to do the random fleet arrangement. For this you need a `randomFleetArrangement` function. The function prototypes of these functions in the header file could be something like this:

   ```
    /* This function parses the command-line input
       arguments in argv and stores the user-provided
          board size (if provided) at the memory
       location pointed to by board_size and the
       maximum number of shots allowed at the memory
       location pointed to by max_shots. Returns 0
       for successful parsing and -1 if there is an
       error in user input.
   */
    int parseArgs(int argc, char **argv, int *
       board_size, int *max_shots);


   /* This function generates the random fleet
      arrangement on blank_board and returns a
      pointer to the new board.
   */
    board * randomFleetArrangement(board * blank_board
       );
   ```

   In this way, think of the other functions that you will need in a sequential manner.

4. You can indicate the fleet arrangement by a matrix whose entries can take five values. 0 would indicate no ship in that cell and a number from $1-4$

would indicate the type of ship. For example, your matrix can look like this:

```
        1 1 0 0 0 0 2
        0 0 0 0 0 0 2
        0 0 3 0 0 0 2
        0 0 3 0 0 0 0
        0 0 3 0 0 0 0
        0 4 4 4 4 0 0
        0 0 0 0 0 0 0
```

Here 1 indicates a submarine, 2 indicates a cruiser, 3 indicates a battleship and 4 indicates an aircraft carrier.

5 Standard Battleship rules

Basic rules for playing a standard two-player game of battleship:

- Both players start by secretly arranging their fleet of ships on their respective boards. Ships may only be arranged horizontally or vertically, may not overlap with each other, and must be fully contained within the board.
- Once all the ships are arranged, players take turns firing shots at the other player's board by announcing the cell number of the opposition player's board they want to hit. The opposition player must announce whether the shot hit any of their ships (i.e. whether that cell was occupied by a ship or not. They may announce this by simply stating ”Hit” or ”Miss”.) Once all the cells occupied by a particular ship are hit, they must announce this to the other player (for example, by stating something like ”You sunk by Destroyer!”.)
- The player who hits all the cells occupied by ships on the other player's board first wins the game.