<u>**Oxo extension- Daniel Davies**</u>

<u>Battleship</u>

The extension task that was completed for this time was a "Battleship" game. The game is a classic board game, in which 2 players take turns to sink each other's ships and therefore win the game. The game consists of 2 boards, X^2 in size (10 traditionally), and a number of ships that the player can place wherever they like on their board, providing they do not overlap, run off the board, and are horizontal/vertical only. Players cannot see each other's boards of course.

Once the boards are initialised, the players take turns in guessing a row and a column to guess where an opponent ship is located. If the guess is a miss (the location they guessed is blank) then the other player has a turn, but if the guess is a hit, the player gets another go in trying to sink the ship that they just hit. (Note- once a game has begun, ships cannot be moved). A player wins when all the opponent's ships have been sunk.

<u>Program</u>

The program is fairly like the traditional game, but with 3 key differences:

-traditionally, the board for each player is a 10x10 grid. In this program, however, due to terminal restrictions, the grid was made 7x7. This also means a game takes a shorter amount of time to play (but still lasts relatively long)

-for the sake of simplicity, the program has a fixed length ship of size 3 (traditionally there are ships of different sizes)

-in addition, since the grid is also now smaller, there are 3 ships rather than the traditional 5

The program works much the same as a traditional game of battleship- however, since the game relies on players not being able to see each other's boards, 2 players playing on the same computer screen, would defeat the object of the game. Therefore I decided to instead create a computer AI that would, as best as possible, imitate what a real player would do. As for the interface, just like in a real game of battleship, the player has 2 boards visible: a board of their guesses (and ship hits) on a board (which represents their attacks on the computer), and their own board, which shows the player's ships, as well as where the computer has guessed.

Initially the player inputs where they would like to place their 3 ships. This is input by specifying a direction (horizontal/vertical placement) followed by a row and a column. After a validity check is passed and ships have been input and recorded by a structure, the AI selects its ships at random (only on the condition that 2 ships must be vertical and one must be horizontal and one ship must be vertical- more for style than anything else. This is also validity checked.). After this, the computer and player take it in turns to guess where each other's ships are. The player simply inputs guesses into a computer like in a real game of battleship- guessing by row and column until a hit is found, at which point this is displayed to the user, and they are given another go. Players are also notified it they have sunken a ship, or, if one of their ships has been sunken.

The AI follows this process as well, but to give the player an element of challenge, the AI tries to play tactically, based on the following rules.

-firstly, the board is divided into 4 separate areas of guessing (roughly equal, but not quite because of the board size)

-in each area the computer will only guess 2 co-ordinates, and then move on to the next area. This ensures that the entire board (close enough anyway) is being guessed at

-the computer starts in "Area 0" (top left). A random co-ordinate is generated. If this is a miss, get another co-ordinate

-if this is a hit however, then the AI methodically guesses the surrounding area (North, East, South, West) for another hit, which reveals the bearing of the ship

-From this point, the AI essentially target locks the ship it has just found. It guesses until a miss is recorded (ship is destroyed) or until the full cycle is complete (indicates the ship was already partially destroyed)

-once the cycle is complete, guess another co-ordinate (in the same area if it had a hit on the first guess, or in another area if it already had 2 guesses in that area)

-carry out this system until the game is complete

This AI formed the majority of the program, and was by far the most logically complex part to complete (mostly because of the number of edge-cases in a game of battleship- but also because of the number of possible inputs.) The AI generally works well and provides a decent challenge for the player. The only compromise to this method of guessing was in the case that the centre of a ship is hit- because the cycle of the AI will hit one end of the ship, but terminate before hitting the other. However luckily, because of targeted area guessing, the remainder of the ship is likely to be hit soon after, and hence not much is changed. As a bonus, this also buys more time for the player. Hence I decided to stick with this method.

The player and AI continue to guess each other's ship positions until all of one sides ships are eliminated and the game is won.

Auto testing has been carried out in number in this project to ensure all functions are working correctly. All functions were tested using auto testing (either directly-by physically calling the function and then testing the effects/output, or indirectly- whereby the already tested functions make up a larger, organised function, and hence testing the smaller functions also test the larger functions indirectly). To run auto testing, simply uncomment the "testing" in main (and also uncomment "Initialise Game" function).

Input to battleship is almost the same as in Oxo, just with a larger grid. For guessing, a row letter and column number is simply input to guess the position on the grid.

While inputting the ships to the board however, the user is also required to enter a "v" or "h" before the column and row to indicate whether they want to place the ship horizontally or vertically.

<u>Misc.</u>

A board size of 7x7 specifically was included since I found this was generally the most suited to the terminal window (especially since 2 are being printed simultaneously side by side). A board size of 7x7 also gives a reasonably times game- not too long, but not too short either.

Initially, only one board (the player board) is printed when inputting where ships are to be placed. Later however, 2 boards are printed side by side- the left-hand side board representing where the player has guessed, the right-hand side where the computer has guessed (and where the player has stored their ships). This was done because in a traditional game of battleship, these are the 2 boards the user has access to.

When inputting ships, the format is something of the sort "ha1". The directional bit (the "h" or "v") specifies if the ship is to be placed horizontal or vertical.

When placing a ship, the base of the ship (first point) is placed at the specified co-ordinate, then the remaining 2 co-ordinates of the ship are placed as follows:

-if horizontal, the remaining 2 co-ordinates are added to the right-hand side of the base

-if vertical, the remaining 2 co-ordinates are added directly above the base

this system of ship input was used as it is generally simple to use/understand, and all possible layout of ships is achievable with this method.

Overall, I think this project was successful- the battleship game is not only functional, but the AI also provides a reasonable challenge for the user. The board style also works well in the given terminal window. More importantly, I believe I learned a great deal about structures and arrays while programming this game.

Questions

While completing this project, I came about some questions I wanted to ask:

You may notice that I used 4 separate structures. There are separate structures for holding a computer/player guess, and the computer/player information itself. These technically could have been combined into one/two structures, but for the sake of clarity, I decided to create 4 separate ones to deal with each aspect of the game. Was this the correct thing to do, or would combining the information into two structures have been better? (for example, a player struct and a computer struct)

I tried my best to keep functions as small as possible, but was just wondering, is there a limit to "too big" for a function?

On this occasion, I didn't include any in-function comments (unless needed to explain a concept I thought was unclear). Is this generally the preferred method, or are in-comment functions worth it?