BATTLESHIPS

UCP assignment 2020

Stephen den Boer
19761257

Table of Contents

1. File descriptions

1.1. Board.c

The board file is responsible for maintaining the board struct. This struct holds information about the size, shape and appearance of the board as well as keeping track of the ships on the board. It contains all the necessary functionality to populate the board with ships, evaluate shots on the board, and display the board.

1.2. Ship.c

The ship file takes care of the ship struct. This struct contains the length, direction and starting coordinate, to allow the ship to be placed on the board in the correct location. It also contains the number of hits the ship has taken in order to determine if the ship has been sunk. The name of the ship is also used in order to identify the ship

1.3. Coord.c

This file contains the functions required to utilise the coord struct. This struct represents a position on the board, storing its horizontal and vertical component. It is used to mark the starting position for ships as well as when obtaining a coordinate to fire at.

1.4. Missile.c

This file manages the missile struct. This struct contains the type of missile, a message explaining how the missile works, and a reference to a function which allows the missile to be executed. The missile is used to fire at the board to try and hit a ship. There are four different types of missiles that hit one or more coordinates on the board. The missile is set up in such a way that it can be stored in a linked list.

1.5. LinkedList.c

The linked list file stores an implementation of the popular linked list datatype. This implementation features a double ended, doubly linked list. What this means is that each item points to the item before and after it and the list keeps track of the beginning and end of the list. The best thing about a linked list is that it can be created without knowing how many items will be added to the list. For this program, the linked list is solely used to store the missiles.

1.6. FileIO.c

This file contains the necessary functionality to read a list of missiles and a board struct from file. The program requires the user to provide a board and missile file when running the program. These files are then read and used to play the game.

1.7. start.c

This file is the starting point for the program. It contains one function which initiates the program execution. It calls FileIO to read the missile and board files, and then calls the menu if both the files were valid.

1.8. Menu.c

This file contains the menu which allows the user to choose how they want to interact with the program. Once the user has selected an option, the relevant function will be called in order to complete that option. Once completed, the user will then be asked to provide another option. This continues until the user chooses to exit.

1.9. GAME.C

This file is in charge of running the game. It features a loop which will display the board as well as: the current missile, the number of remaining missiles, and will ask the user for the next coordinate to target with the missile. This continues until the missiles run out or all the ships are sunk.

1.10. CREATEFILE.C

This file manages the capability for creating new board and missile files. This capability walks the user through the steps of entering the information that makes up these files. The user's entries are checked to make sure only valid information is added to file.

1.11. USERINTERFACE.C

The contents of this file manage input from the user. It performs multiple checks on the data received so that the calling code can be assured that it will return a valid result. It also provides methods for basic string manipulation, such as converting to uppercase and joining two strings.

1.12. TESTFILES

1.12.1. INPUT FILES

There are numerous input files provide for testing the program in different ways. The files prefixed with 'in' are input board files while those prefixed with 'mi' are input missile files. The first 2 board files, and the first missile file are valid input files, while the rest are invalid input files.

1.12.2. USER INPUT FILES

There is also a number of user input files provided. These are the files prefixed with 'test'. These files hold sample input which can be re-directed to the program instead of manually entering data. The number after test represents what menu item they are testing. For example, test1 tests the first menu option.

2. INTERACTION BETWEEN SHIPS AND BOARD

The board contains 2 arrays, one array which contains the characters to be printed and the other contains either a pointer to a ship, or a null reference. The reason for this was to easily determine if a given coordinate contained a ship, and to update that ship if necessary. In order for this to work, a reference to the same ship needed to be stored in every position the ship occupied in the board. This means the ships needed to calculate the indices where they would be stored in the board and then upon adding the ship to the board, the ship would need to occupy each of these indices in the ship array.

This method made it easy to determine if a given tile contained a ship. It also provided the correct ship to modify, making it easy to check if the ship was sunk, by comparing the number of hits to its length. However, this made freeing the board a little bit more complicated since each reference to the ship in the array had to be set to null, otherwise a free on the same ship would be attempted multiple times.

3. USER GUIDE

3.1. INPUT FILES

Two valid input board files:

```
1 5,4
2 d4 E 3 Sinking Sailboat
3 A1 n 3 defiant destroyer
4 C2 W 2 MHS Lucetania
```

```
1 12,12
2 D4 E 3 HMS endeavour
3 A1 N 10 The Titanic
4 C2 W 2 secret sub
```

A valid missile input file:

```
1 single
2 splash
3 single
4 V-line
5 h-line
6 single
```

Note that all text is case insensitive and there is no limit on the number of ships, provided they don't cross over.

3.2. Running the program

The program is run with the following command:

```
./battleships boardFile missileFile
```

Where boardFile and missileFile are valid missile and board files as demonstrated above.

Upon running this command, the game will display a menu:

```
Please enter the integer that corresponds to your choice
1. Play the game
2. List missiles
3. create Board file
4. create Missile File
0. Exit
```

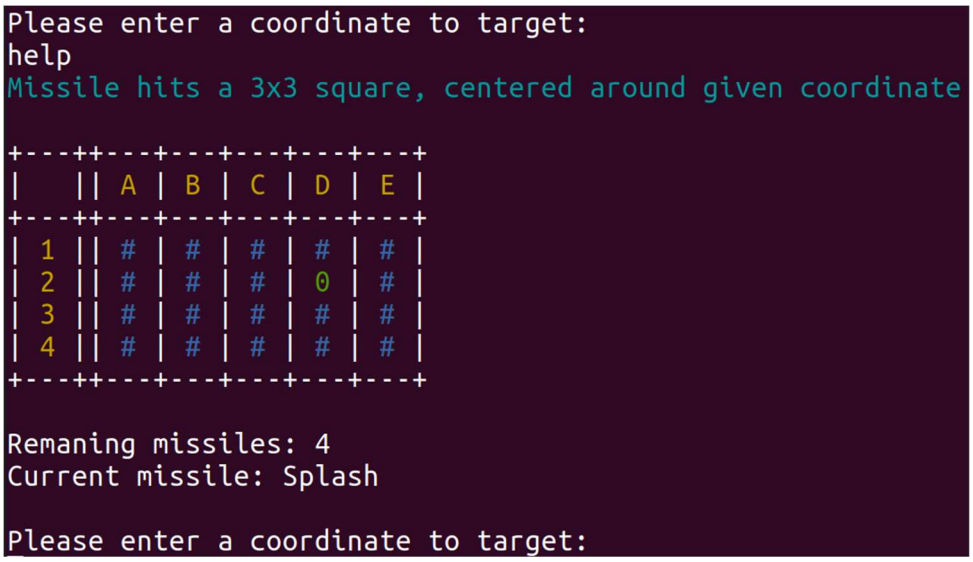
The user should then enter the single digit corresponding with their choice, followed by pressing enter.

3.3. Interacting with the game

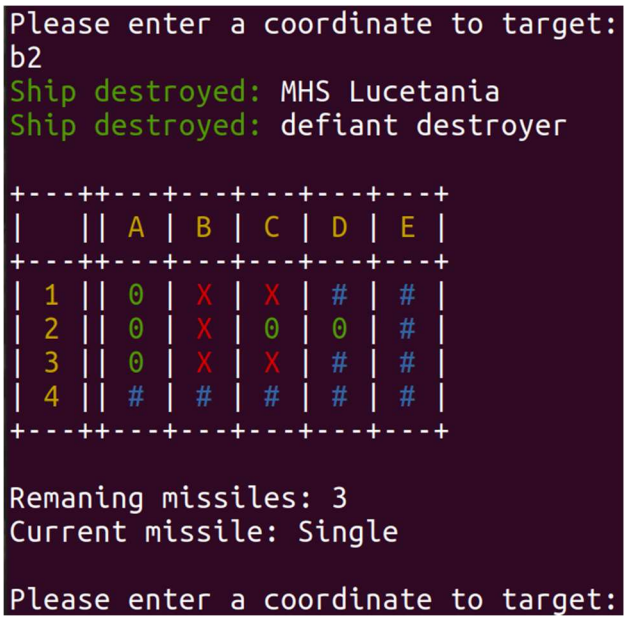
Upon choosing to play the game, the user will be presented with a view of the board and asked to enter a coordinate:

The coordinate should be a letter from the horizontal axis followed by a number on the vertical axis which should be followed by the enter key. The character is case insensitive. After entering the coordinate, the board will be updated to reflect your shot. A green 0 is a hit and a red X is a miss.

You also have the option of typing ‘help’ as a coordinate. This too is case insensitive. Upon entering help, the program will display information about how the current missile works and re-display the board.



Upon sinking a ship, a message will be displayed, to notify you:



Upon sinking all the ships, you will be congratulated and prompted with the choice to play again or re-visit the main menu.

```
Ship destroyed: Sinking Sailboat
Congratulations! You sunk all the ships

+---++---+---+---+---+---+
|   || A | B | C | D | E |
+---++---+---+---+---+---+
| 1 || 0 | X | X | X | # |
| 2 || 0 | X | 0 | 0 | # |
| 3 || 0 | X | X | X | # |
| 4 || X | 0 | 0 | 0 | X |
+---++---+---+---+---+---+

Please enter the integer that corresponds to your choice
1. Play again
0. Back to Main menu
```

Alternatively, if you run out of missiles, you will be shown the message: “Game Over” and prompted with the same menu.

```
Game over.

Please enter the integer that corresponds to your choice
1. Play again
0. Back to Main menu
```