Hello Owen and welcome to Sinker.

Sinker is a battleship game played in a Java dialog box.

I don’t have any algorithms or flowcharts because it was started as an exercise to learn gitHub with a mate of mine. It was designed in an intentionally shortsighted way, never thinking about anything except the current problem.

The process was essentially like this:

We began with the first playable version which was a single line char array with a char randomly inserted into it:

-,-,-,V,-,-

There was no hidden info and as soon as you input the index in the array occupied by the ‘V’ you won the game. Then we decided the next two or three features we would add on a whiteboard and worked out how to implement those.

And that’s how we continued until the version you have.

The basic structure is as follows:

1. Run the project.
2. Generate 2 grids; one displayed to the player (p) and one hidden (g).
3. Create ship objects.
4. Spawn ship objects randomly onto grid g.
5. Player inputs number of space they are targeting onto grid p.
6. Game loop checks whether the space contains a ship segment.
   1. If ship segment is in selected space mark ‘H’ on grid p.
   2. If space is empty mark ‘X’ on grid p.
7. Every time a ‘H’ is generated deduct a hitpoint from the relevant ship (b,c,d for Battleship, Cruiser, Destroyer). When hitpoints for a particular ship reach zero display a “You sunk an enemy …..” message.
8. When all ships are sunk game ends with a tally of the shots taken.

The central aspect the game is built around is the grid, which is a one dimensional char array (not a 2d one because of the deliberate shortsightedness). There is a variable in the main class called gridSize which is user definable. The Grid Class makes a square grid from that value and adds extra indexes containing ‘\n’. In effect the grid is analogous to an old CRT raster scan, moving to a new line at specific intervals.

What that means is that there is a disconnect between the grid as displayed to the user, and the grid used internally for calculations.

Here is an example using gridSize 4

Displayed grid Internal Grid

*Column Offset*

*containing ‘\n’*

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 0 | 1 | 2 | 3 | 4 |  |
| 5 | 6 | 7 | 8 | 5 | 6 | 7 | 8 | 9 | +1 |
| 9 | 10 | 11 | 12 | 10 | 11 | 12 | 13 | 14 | +2 |
| 13 | 14 | 15 | 16 | 15 | 16 | 17 | 18 | 19 | +3 |

Shots are input, and starting ship positions are generated as if the grid on the left is the grid the computer uses. Both the shots and ship positions are actually calculated using the grid on the right.

There is a method towards the end of the main class called ***transform()***, which takes the position value as given in the left grid and calculates the offset needed to map it to the equivalent space in the right grid. A description of it in action is in the comment beside the ***transform()*** method.

And that’s probably enough of an overview. The next step is a GUI with boxes that the user clicks on, after that an AI for some competition and then a volley mode where each side gets as many shots as they have ships on the field. And next year when I know something about teh internetz, maybe a two player mode.

* Cormac

PS - There isn’t any exception handling on the shot input so if you put in a + or a ^ or anything which isn’t an integer the game will crash. It won’t matter when I move to a GUI so I didn’t bother with it.