

# BattleShip

## Matthew's Stats

Time Taken: A weekend

Lines Of Code: ~1500

Files: 41

## What to Submit

- A CMakeLists.txt that can compile your program into an **executable named BattleShip**
- The **files** including any subfolders if you have them, that make up your project.

## Problem Description

You will be implementing the game of [BattleShip](#). The game starts with each player secretly placing their ships on their board. A ship can be placed either horizontally or vertically on their board. Once the players have finished placing their ships, they take turns guessing locations on their opponent's board to fire. Their opponent announces whether the shot hits or misses their ship and if it is the final hit on their ship, the fact that that shot destroyed their ship. You can [play a version of BattleShip here](#) and I recommend that you do so if you haven't played the game before.

## Input

Input will come from 3 locations: the command line, files, and standard input

### Command Line

- Argument 1
  - Required
  - The path to the configuration file for this game
    - This value will always be valid
- Argument 2
  - Optional
  - An integer to be used as the seed for the random number generator
    - This value will always be valid
  - If the seed is not given the current time should be used
    - This has already been implemented for you

## Configuration File

The configuration file specifies

- The size of the board
- The number of ships to be placed on the board
- The character used to represent each ship
- The size of each ship

The format of the file looks like

- Number of rows on the board
- Number of columns on the board
- Number of ships to be placed
- Ship1\_Character Ship1\_Size
- Ship2\_Character Ship2\_Size
- ...

For example, the default configuration file looks like

```
10
10
5
C 5
B 4
D 3
S 3
P 2
```

The contents of this file will always be valid.

## Standard Input (Keyboard)

This is how the user will specify where to place their ships and what location on the board to fire at. Input will always be valid but it is recommended that you do some input validation to make your program less fragile.

## Output

The output is a bit too complicated to specify here what it should look like. I have given you a compiled version of my program on Kodethon that can be run on Kodethon. Please play around with it to get an idea of what output should look like.

# Game Modes

The game can be played in 3 different modes

1. Human vs Human
2. Human vs AI
3. AI vs AI

## Setting Up The Game

### Initializing the Random Number Generator

If a seed is given you should initialize the random number generator found in AiPlayer exactly once with the given value and **BEFORE** you make any calls to random functions. This is extremely important as different seeds cause the random number generator to produce different random values which would lead to us having different results even if the rest of your code is 100% correct. You will also have to make the same number of calls to the random generator and in the same order to match my output so please read any instructions involving random very carefully.

### Human

A human will first be asked for their name and then will be asked to place their ships. Ships are placed in ASCII order based on the character used to represent the ship.

For each ship

- The player's board should be displayed
- The player should be asked if they want to place the ship horizontally or vertically
  - H or h represents horizontal and V or v represents vertical
- They will then be asked for the starting coordinate they want to place the ship at
  - This will be the leftmost point if the ship is placed horizontally
  - This will be the topmost point if the ship is placed vertically
- If the ship can be legally placed at the location it should be placed there but if it can't this process should be repeated again for this ship until a legal placement is chosen
  - A placement is legal if
    - The ship can fit there without going off the board
    - The ship does not overlap with any of the ships that have already been placed

## AI

Ships are placed in ASCII order based on the character used to represent the ship.

For each ship

1. The AI's board should be displayed
2. The AI randomly determines if the ship should be placed horizontally or vertically
3. The AI chooses a random starting row and column such that the ship could be placed there without going off the board
4. If the placement doesn't overlap with any previously placed ship the ship should be placed there, but if not the process begins again at step 2

I have already implemented this for you so please use the code that is there. You might have to make some small tweaks to it if you change variable names or the structure of the class but the number of calls and the ordering of those calls should be very apparent and should not be changed.

## AI's

### Naming

The first AI created is named AI 1. The second AI created is named AI 2.

### Cheating AI

The cheating AI plays a perfect game of BattleShip. It never misses and always hits the opponents ship. The AI shoots at the ships in sequence, starting from the upper leftmost corner and working its way down to the lower rightmost corner. For example, if the opponent's board looked like

```
  0 1 2 3 4 5
0 * * * * *
1 A * * * *
2 A * * * *
3 A C * * *
4 * C * * *
5 * C B B B *
```

The firing sequence would be (1,0), (2,0), (3,0), (3,1), (4,1), (5,1), (5,2), (5,3), (5,4).

## Random AI

The random AI randomly guesses locations to fire at on the board. It never guesses the same location twice. To implement this you should

- Generate a vector of all the locations to fire at starting from the upper leftmost corner and ending at the bottom right most corner
  - You should do this when you create your RandomAI
- Randomly select and remove an element from this list as your firing location
  - If you look in Utility.h that I've given you there is a function there that will help you randomly select an element. It gives you back an iterator to the value
  - After you select the value you can remove it from the vector by using the vector's erase method
    - Don't forget to copy the value that is pointed to by the iterator before deleting it

## Hunt Destroy AI

This AI operates in 2 distinct modes: Hunt and Destroy

### Hunt Mode

In Hunt mode, the AI behaves like the Random AI and randomly guess at locations to shoot. As soon as it gets a hit it switches to Destroy mode.

### Destroy Mode

In Destroy mode, the AI will shoot at all locations around the oldest location hit. The AI will fire at the surrounding locations in this order: left, up, right, then down. If any hits are scored they are added to the list of locations to fire at. You return to Hunt mode once all of these locations have been fired at.

## Example

Read from top to bottom, right to left.

```
0 1 2 3 4 5
0 * * * * *
1 * * * * *
2 * * * B B B
3 * * * * *
4 * * * * *
5 * * * * *
```

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | * | * | * |
| 2 | * | * | * | X | B | B |
| 3 | * | * | * | * | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | * | * | * |
| 2 | * | * | O | X | B | B |
| 3 | * | * | * | * | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | * | * |
| 2 | * | * | O | X | B | B |
| 3 | * | * | * | * | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | * | * |
| 2 | * | * | O | X | X | B |
| 3 | * | * | * | * | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | * | * |
| 2 | * | * | O | X | X | B |
| 3 | * | * | * | O | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | * | * |
| 2 | * | * | O | X | X | B |
| 3 | * | * | * | O | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | O | * |
| 2 | * | * | O | X | X | B |
| 3 | * | * | * | O | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | O | * |
| 2 | * | * | O | X | X | X |
| 3 | * | * | * | O | * | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | O | * |
| 2 | * | * | O | X | X | X |
| 3 | * | * | * | O | O | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | O | O |
| 2 | * | * | O | X | X | X |
| 3 | * | * | * | O | O | * |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

|   | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | * | * | * | * | * | * |
| 1 | * | * | * | O | O | O |
| 2 | * | * | O | X | X | X |
| 3 | * | * | * | O | O | O |
| 4 | * | * | * | * | * | * |
| 5 | * | * | * | * | * | * |

Now back to Hunt Mode.

# Handout

To help get you started I've given you all of my header files. You are free to use or not use them as you see fit. I've also provided some code for dealing with random numbers in Utility.h. Please make use of it in your answer. I have also implemented ship placement for AIs. It is AiPlayer.cpp. AiPlayer.h and AiPlayer.cpp have the random number generator that you should be using. It is called randomNumberGenerator.

## Hints and Musings

- The most time-consuming part of the project is getting the game setup. Once all the boards and players are in place it is quite straightforward to implement the gameplay logic
- Play the executable that I have given you. It will help you figure out what output should look like
- Good use of inheritance is key in this problem to make it manageable
- Think through the project a bit before you start coding.
  - Figure out what things are made up of what.
  - Solve the problem from the top down. Start board and narrow things down only as you need to. Believe that there is a function that does exactly what you need it to do and call it. Then come back and make that function
- A map is very useful to keep track of how many hits a ship has left
- What you store and what you display do **NOT** have to be the same thing
  - You don't have to have a vector of strings to represent the board
  - You don't have to store the spaces in between cells
  - You could have an easy to work with model of the game and then figure out how to print it to the screen in the way that I want
- It is not actually necessary for each player to have 2 boards. You could use your opponent's board to figure out what to display for your "Firing Board"