CSE 202: Implementation Report

Martakis, Alex
amartakis@ucsd.edu A59012834

Sharan, Mayank
msharan@ucsd.edu A59012192

Choudhary, Twinkle
twchoudhary@ucsd.edu A59016512

Gupta, Satvik
sag005@ucsd.edu A59012252

March 17, 2023

1 Introduction

Our project is to model the game of battleship and solve algorithmic problems associated with it. In this document, we present implementation based analysis of the algorithms we propose. We include a refresher of the game and our modeling below.

Battleship is played between 2 players. The objective of the game is to sink all ships of your opponent before they are able to sink yours. The gameplay has the following phases:

1. **Setup:** Both players arrange the ships they have on their grid.
2. **Guessing:** Both players take turns guessing a grid position of the opponent each turn
   - In this phase, if a guess is where an opponent ship is present, it is a "hit".
   - otherwise, it is a "miss".
   - A ship is sunk if all grids it is located in, are hit.
   - The player who sinks all ships of the opponent first wins.

Some key components of the game to be modeled are:

1. The board
2. The ships: count and length
3. The arrangement of ships
4. Player guesses and their outcome ("hit" or "miss")

In Section 2, we define the basic variables required to model the game which will be used in multiple algorithmic problems. We also discuss the restrictions we impose to limit the complexity of the problems we tackle.

2 Game Modeling

2.1 Board Size

A typical game of battleship is played on a square board of size 10x10. However, to define more general problems and algorithms we will consider rectangular boards of arbitrary size $m \times n$, where m is the number of rows and n is the number of columns.

Without loss of generality we will assume $n \geq m$. This holds as any board with more row than columns can be rotated by 90° to get a board with more columns than rows. Thus, all configurations and actions have a rotational mapping.

Each grid position on the board will be indexed using a tuple of the form (x, y), where x is the row number and y is the column number. Further, $1 \leq x \leq m$ and $1 \leq y \leq n$.

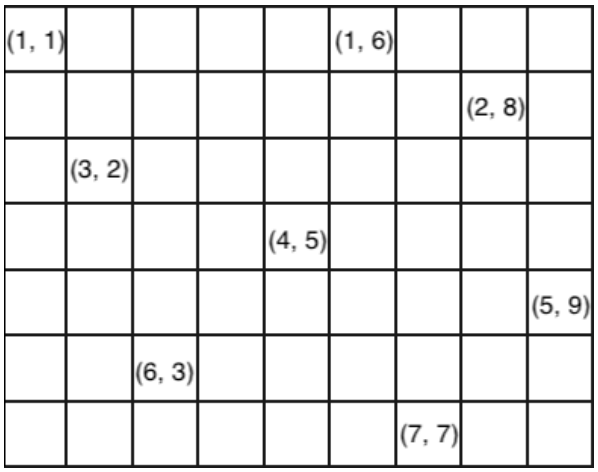

Figure 1: A rectangular board of size 7 x 9 with grid positions illustrated

2.2 Ships

A standard battleship game has five ships of length 2, 3, 3, 4, and 5. In the spirit of generalization, we will take the number of ships and their sizes as parameters. We will denote the number of ships as n_s. We will restrict each ship to be 1 grid wide, so the only dimension they have is length.

$1 \leq n_s \leq m$, as we want there to be at least one ship. The upper limit to ensure that if needed, each ship can be put on separate rows or columns. The number of ships also contributes to the nature of the game. A sparse setup will likely have a very different approach than a dense one.

The length of the ships would be provided as an array $shipLen$ of length n_s, indexed from 1 to n_s. $1 \leq shipLen[i] \leq m$ and $shipLen[i] \in \mathbb{Z}^+ \; \forall 1 \leq i \leq n_s$. This is to ensure that every ship can fit within any row or column it is placed on.

2.3 Arrangement of Ships

For any given ship with identifier i its location on the grid for player p is defined as a pair of tuples $(x^p_{is}, y^p_{is}), (x^p_{ie}, y^p_{ie})$, both valid coordinates on the board as defined in Section 2.1. For simplicity, we order the tuples so that $x^p_{is} \leq x^p_{ie}$ and $y^p_{is} \leq y^p_{ie}$. Here, $x^p_{is} = x^p_{ie}$ for a placement along a row or $y^p_{is} = y^p_{ie}$ for a placement along a column. We do not allow diagonal placement of ships as per the rules of the game. In case of a row placement $y^p_{ie} - y^p_{is} + 1 = shipLen[i]$ and in case of a column placement $x^p_{ie} - x^p_{is} + 1 = shipLen[i]$. These tuples for each ship are stored in lists of tuples $shipS_p$ containing all (x^p_{is}, y^p_{is}) tuples and $shipE_p$ containing all (x^p_{ie}, y^p_{ie}) tuples. Both arrays are indexed by the ship identifier i.

Ships are not allowed to overlap so there does **NOT** exists any tuple (x, y) such that for any 2 ships i and j the following conditions are all true:

$$x^p_{is} \leq x \leq x^p_{ie}$$
$$x^p_{js} \leq x \leq x^p_{je}$$
$$y^p_{is} \leq y \leq y^p_{ie}$$
$$y^p_{js} \leq y \leq y^p_{je}$$

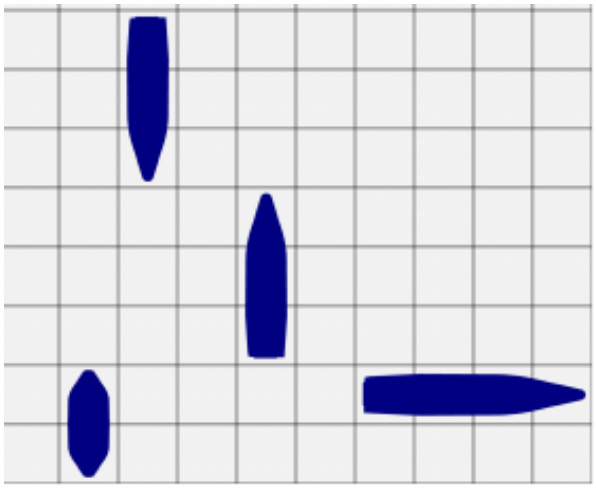

Figure 2: A sample arrangement of ships

2.4 Guess outcomes

We maintain 2 two-dimensional matrices to represent the guess status. For players p and q, these matrices are $Guess_p$ and $Guess_q$. Both matrices are of size $m \times n$ in correspondence to the board and are indexed starting from 1 so that the position tuples can be used as indices. $Guess_p[(x, y)]$ will provide the status on player q's grid of the guess by p at location (x, y) and vice versa.

Both matrices are initialized with all "U"s to indicate unexplored. As the game progresses in phase 2 the outcomes of all guesses by player p are updated in $Guess_p$ and by player q are updated in $Guess_q$. The value is updated to "H" in the event of a hit and to "M" in the event of a miss.

Both players have access to both matrices at all times as the information in them is available to both of them as the game progresses. We will not be storing the order of the guesses as it introduces storage complexity without adding any significant information and is not a reasonably expected for human players to be able to track.

3 Implementation Results

3.1 Finding the first hit

In our algorithm solutions report we presented 3 algorithms for finding the first hit,

- Random
  - We guess randomly till we find the first hit
- Diagonal
  - We guess along the the diagonals mutually exclusive for the largest unsunk ship
  - Order of guesses is in decreasing order of number of possible arrangements till the first hit
- PDF
  - We compute a pdf based on possible arrangement of all unsunk ships
  - We guess a location with the highest count
  - If we miss we update counts for affected neighbors and continue till the first hit

The runtime is expected to increase in the order Random $<$ Diagonal $<$ PDF and the number of guesses required is supposed to be lesser for Diagonal and PDF when compared to random. However, it is not clear if one will do better in all scenarios.

3.1.1 Experiments

For this algorithmic problem we ran experiments across 3 different variables to test the number of guesses taken and the runtime. The variables were as follows

- Board Dimension: [10, 10], [10, 13], [14, 16]
- Ship Lengths: [2, 3], [2, 3, 4, 5], [7, 8, 9]
- Initial Guesses Proportion: 0, 0.2, 0.3

Notice the board sizes are increasing, so is the sum of the ship lengths. The initial guess proportion is the percentage of grid cells that have a guess before we start our algorithm. This initialization is done such that there are no partially sunk ships and that there is at least 1 unsunk ship to ensure the initial requirements of the algorithm are met.

We ran each of the 27 configurations across 10000 different random seeds to generate different initial configuration of ships and different initial guesses. The number of arrangements are very high so to ensure that the metrics we report are close to the true expectation the number of samples needed to be high. We observed that 100000 samples provided slightly better accuracy but due to time constraints we were not able to run that amount for all configurations.

The experiments were run on a machine with the following configuration.

Programming Language: Python 3 using Jupyter

Machine: Macbook Pro 14"; M1 processor with 8 cores; 16 GB RAM

3.1.2 Results and Discussion

Number of Guesses

Since, only expected outcomes are of our primary concern here we will report averages. The results are shown in Table 1. As expected diagonal and PDF perform significantly better than random in all scenarios. The gap is larger at higher board sizes as expected.

The comparison between diagonal and PDF is quite interesting. In scenarios where we start from a clean board it is comparable to sometimes even slightly better than PDF. Diagonal performs marginally better in scenarios with smaller ships. This indicates that in such scenarios it is able to reduce the number of guesses right away and guess as per the largest ship size thus charting roughly similar priority order as the PDF without having to go through as many guesses. The PDF on the other hand is limited by the fact that every miss gives it very little information.

This is completely different once we start with some initial guesses already made. The additional complexity that is introduces in this scenario is modeled much better by the PDF algorithm and that is clearly visible in the results. In these settings the PDF does better by 0.5 to 1 guesses on average typically, to even 1.5 guesses better than Diagonal in the setting with 0.3 initialization, [2 3] ship lengths and 14 x 16 board size. In this case it improves on the random algorithm by half. The last 3 rows show a 3-4x improvement over random in an impressive scenario where diagonal and PDF take 1 to 2 guesses on average to find the next hit.

Note: In the early days of the project we were planning to work with the random and diagonal algorithms. Even the diagonal algorithm did not use counts for each cell to order them to improve the expected number of guesses. However, we noticed we were not doing much better than the random in cases where diagonal would have ships located on the off diagonal. At this point the idea came to improve it by adding counts which was verified using similar analysis as is presented here but on a much smaller scale. We then further extended the idea that the count from the largest ship can be done for all cells and then added to similar counts for all other cells. It seemed to be the idea that would do best. Thus, based on the visual examination of the results from our presentation we got ideas to further improve and even introduce a new algorithm. This was a valuable way for us to improve our methods as the mathematics to find comparisons and guarantees was often intractable for us. So we used empirical analysis to test our theories instead.

Table 1: Average number of guesses taken to find the first hit by different algorithms (10,000 samples)

Initial Proportion	Ship Lengths	Board Dimensions	Random	Diagonal	PDF
0.0	[2 3]	[10 10]	16.94	11.73	11.79
0.0	[2 3]	[10 13]	21.87	15.32	15.39
0.0	[2 3]	[14 16]	37.72	26.41	26.39
0.0	[2 3 4 5]	[10 10]	6.66	4.79	4.72
0.0	[2 3 4 5]	[10 13]	8.62	6.15	6.16
0.0	[2 3 4 5]	[14 16]	15.09	10.63	10.67
0.0	[7 8 9]	[10 10]	4.08	3.35	3.31
0.0	[7 8 9]	[10 13]	5.22	3.67	3.6
0.0	[7 8 9]	[14 16]	8.96	5.67	5.62
0.2	[2 3]	[10 10]	13.88	8.4	7.85
0.2	[2 3]	[10 13]	18.49	10.82	10.27
0.2	[2 3]	[14 16]	31.14	18.1	17.04
0.2	[2 3 4 5]	[10 10]	5.62	3.24	2.81
0.2	[2 3 4 5]	[10 13]	7.28	4.02	3.51
0.2	[2 3 4 5]	[14 16]	12.18	6.41	5.65
0.2	[7 8 9]	[10 10]	3.47	1.75	1.42
0.2	[7 8 9]	[10 13]	4.39	1.91	1.5
0.2	[7 8 9]	[14 16]	7.46	2.65	2.0
0.3	[2 3]	[10 10]	12.91	7.16	6.48
0.3	[2 3]	[10 13]	16.75	9.15	8.29
0.3	[2 3]	[14 16]	28.54	15.08	13.61
0.3	[2 3 4 5]	[10 10]	5.12	2.64	2.25
0.3	[2 3 4 5]	[10 13]	6.58	3.2	2.67
0.3	[2 3 4 5]	[14 16]	11.1	4.9	4.08
0.3	[7 8 9]	[10 10]	3.19	1.41	1.16
0.3	[7 8 9]	[10 13]	3.93	1.49	1.2
0.3	[7 8 9]	[14 16]	6.59	1.85	1.44

Runtime

The results for runtime of the algorithms are shown in Table 2. As expected the runtime is highest for PDF followed by Diagonal and the least for random across all scenarios.

Interesting observation is that in sets of 3 rows it can be seen that the runtime for all algorithms is linearly dependent on the board size. The runtime of the random algorithm does not vary a lot as higher occupancy board or larges ships reduces runtime only linearly. However for Diagonal and PDF as the initial number of guesses increases the runtime decreases quite significantly. Filling up 20 % of the board reduces the runtime to two-thirds and 30 % reduces to almost half. This is likely due to the fact that they both use the information from these initial guesses to better order the guesses they will make not only reducing the number of guesses required but also the guesses that are available as a part of the set.

In Figure 3, we illustrate a sample with the initial board and the guesses made by the different algorithms to get the first hit.

3.2 Sink the ship

We will examine 2 algorithms presented in our algorithm solutions to sink the ship given a first hit,

- Brute Force
  - Find a neighboring cell that gives a hit
  - Continue exploring in that direction till miss
  - Go the opposite direction

Table 2: Average runtime (in μs) taken to find the first hit by different algorithms (10,000 samples)

Initial Proportion	Ship Lengths	Board Dimensions	Random	Diagonal	PDF
0.0	[2 3]	[10 10]	36	380	2281
0.0	[2 3]	[10 13]	48	495	3035
0.0	[2 3]	[14 16]	76	876	5475
0.0	[2 3 4 5]	[10 10]	28	323	2962
0.0	[2 3 4 5]	[10 13]	35	424	4006
0.0	[2 3 4 5]	[14 16]	52	753	7503
0.0	[7 8 9]	[10 10]	25	258	3115
0.0	[7 8 9]	[10 13]	30	339	4330
0.0	[7 8 9]	[14 16]	44	643	9046
0.2	[2 3]	[10 10]	34	286	1496
0.2	[2 3]	[10 13]	43	371	1992
0.2	[2 3]	[14 16]	58	610	3401
0.2	[2 3 4 5]	[10 10]	24	221	1775
0.2	[2 3 4 5]	[10 13]	29	285	2355
0.2	[2 3 4 5]	[14 16]	43	484	4168
0.2	[7 8 9]	[10 10]	27	173	1682
0.2	[7 8 9]	[10 13]	32	214	2212
0.2	[7 8 9]	[14 16]	44	358	3961
0.3	[2 3]	[10 10]	30	241	1173
0.3	[2 3]	[10 13]	36	307	1531
0.3	[2 3]	[14 16]	56	516	2653
0.3	[2 3 4 5]	[10 10]	24	185	1394
0.3	[2 3 4 5]	[10 13]	31	236	1825
0.3	[2 3 4 5]	[14 16]	38	386	3121
0.3	[7 8 9]	[10 10]	21	137	1308
0.3	[7 8 9]	[10 13]	25	167	1679
0.3	[7 8 9]	[14 16]	35	269	2858

  - If ship not sunk try the same process in the perpendicular direction

- Smart sink
  - Check each direction with a cell away by half the distance of the smallest remaining ship
  - If hit pursue the direction and complete till you miss
  - Then try the opposite direction till you miss
  - Do the same process in the perpendicular direction

The runtime is expected to be higher for the smart sink, however it is not clear which one would require lesser number of guesses. The smart sink seems to check more places but has the advantage of discovering new ships in its search which may help when integrated.

3.2.1 Experiments

For this problem as well we ran experiments across 3 different variables to test the number of guesses taken and the runtime. The variables were as follows

- Board Dimension: [10, 10], [10, 13], [14, 16]
- Ship Lengths: [2, 3], [2, 3, 4, 5], [7, 8, 9]
- Initial Guesses Proportion: 0, 0.2, 0.3

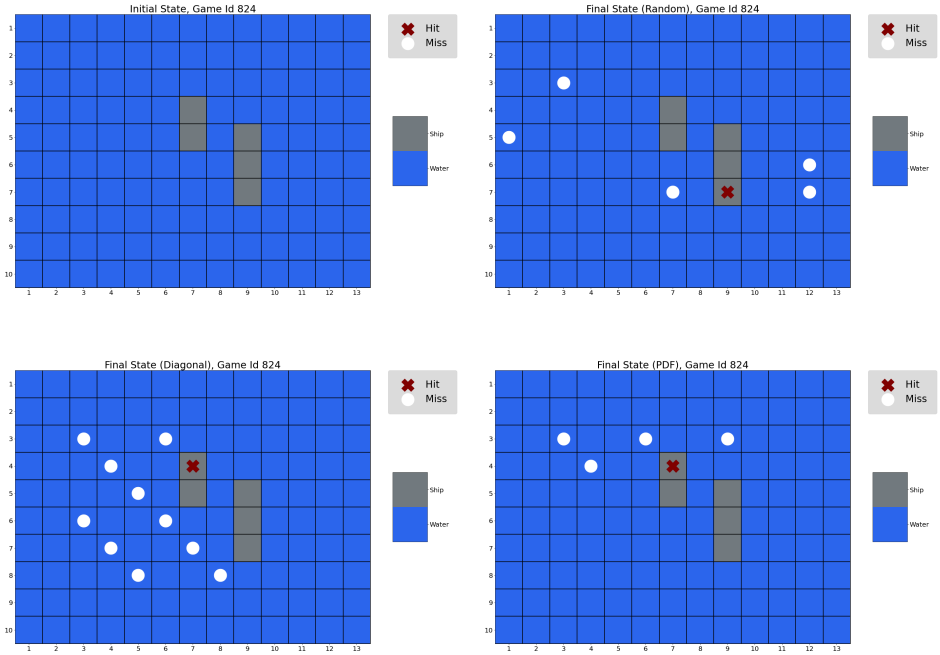

Figure 3: Illustrating the find first hit algorithms. Row 1, L to R: The start setting, the guesses by Random; Row 2, L to R: the guesses by Diagonal, the guesses by PDF;

These are the same set of parameters as used by the first hit algorithms for consistency. We from the initialization added a hit to one of the unsunk ships randomly before giving it as input to the sink the ship algorithms.

We ran each of the 27 configurations across 10000 different random seeds to generate different initial configuration of ships and different initial guesses. The number of arrangements are very high so to ensure that the metrics we report are close to the true expectation the number of samples needed to be high.

The experiments were run on a machine with the following configuration.

Programming Language: Python 3 using Jupyter

Machine: Macbook Pro 14"; M1 processor with 8 cores; 16 GB RAM

3.2.2 Results and Discussion

Number of Guesses

The results are shown in Table 3. We had expected the smart sink to have higher number of guesses but across the board higher by 1 was a bit of a surprise to us. This is due to the factors that sinking the ship is a relatively straightforward problem and might not require complex approaches. We however held out hope that smart sink would provide an advantage in the complete game. More on this in Section 3.3.

The number of guesses as mentioned in our analysis and as expected is directly correlated to the ship sizes. This is expected as the only extra guesses required are to determine the direction and you may go in the wrong direction for a limited number of steps only.

The number of guesses decreases for both algorithms as the proportion of the board filled up increases. It is interesting that the decrease is more for the smart sink. Since it looks farther first it is more likely to benefit from the misses.

Note: Smart sink has an interesting feature that if 2 ships are arranged end to end or in an L and we

Table 3: Average number of guesses taken to sink the ship by different algorithms (10,000 samples)

Initial Proportion	Ship Lengths	Board Dimensions	Brute	Smart
0.0	[2 3]	[10 10]	3.16	4.31
0.0	[2 3]	[10 13]	3.18	4.32
0.0	[2 3]	[14 16]	3.22	4.37
0.0	[2 3 4 5]	[10 10]	4.46	6.11
0.0	[2 3 4 5]	[10 13]	4.58	6.11
0.0	[2 3 4 5]	[14 16]	4.65	6.06
0.0	[7 8 9]	[10 10]	8.29	9.34
0.0	[7 8 9]	[10 13]	8.75	10.0
0.0	[7 8 9]	[14 16]	8.8	10.37
0.2	[2 3]	[10 10]	2.94	3.75
0.2	[2 3]	[10 13]	2.99	3.75
0.2	[2 3]	[14 16]	3.01	3.82
0.2	[2 3 4 5]	[10 10]	4.4	5.75
0.2	[2 3 4 5]	[10 13]	4.5	5.7
0.2	[2 3 4 5]	[14 16]	4.58	5.68
0.2	[7 8 9]	[10 10]	8.17	9.16
0.2	[7 8 9]	[10 13]	8.66	9.54
0.2	[7 8 9]	[14 16]	8.67	9.78
0.3	[2 3]	[10 10]	2.81	3.39
0.3	[2 3]	[10 13]	2.84	3.4
0.3	[2 3]	[14 16]	2.88	3.47
0.3	[2 3 4 5]	[10 10]	4.35	5.5
0.3	[2 3 4 5]	[10 13]	4.45	5.44
0.3	[2 3 4 5]	[14 16]	4.52	5.43
0.3	[7 8 9]	[10 10]	8.13	8.97
0.3	[7 8 9]	[10 13]	8.57	9.3
0.3	[7 8 9]	[14 16]	8.59	9.5

give it a start point from where both lie within the up down left right neighborhood it will sink both ships promptly.

Runtime

The results for runtime of the algorithms are shown in Table 4. As expected the runtime is higher for Smart sink than the brute force.

It is easy to confirm that the runtime is independent of the board size. However, the key factor it depends on is the average ship size. This is not quite linear as the actual dependence is on the number of guesses which are the ship length and a constant of the directions that did not work. Since that constant does not scale with the ship size we see what looks like a slower than linear trend in runtime which in fact is a linear increase of one component and the other staying constant.

As expected the addition of random initial guesses does not affect the runtime in a significant way. As the number of guesses goes down very little so does the runtime.

In Figure 4, we illustrate a sample with the initial board containing the first hit and the guesses made by the different algorithms to sink the ship after a first hit.

3.3 Complete Game

We simulated a complete game from the start on a standard battleship board setup. Board size 10 x 10 and ship lengths [2, 3, 3, 4, 5]. We used all combinations of first hit and sink the ship algorithms to play the games to completion. This yielded in 6 different bots, Rand-Brute, Rand-Smart, Diag-Brute, Diag-Smart, PDF-Brute, and PDF-Smart. Each of these was tasked with taking the same initial board and sinking all the ships and we counted the number of guesses it took to do that. This was run over

Table 4: Average runtime (in μs) taken to sink the ship by different algorithms (10,000 samples)

Initial Proportion	Ship Lengths	Board Dimensions	Brute	Smart
0.0	[2 3]	[10 10]	97	184
0.0	[2 3]	[10 13]	98	184
0.0	[2 3]	[14 16]	98	183
0.0	[2 3 4 5]	[10 10]	131	238
0.0	[2 3 4 5]	[10 13]	134	238
0.0	[2 3 4 5]	[14 16]	134	233
0.0	[7 8 9]	[10 10]	236	331
0.0	[7 8 9]	[10 13]	251	347
0.0	[7 8 9]	[14 16]	245	348
0.2	[2 3]	[10 10]	97	180
0.2	[2 3]	[10 13]	98	179
0.2	[2 3]	[14 16]	97	177
0.2	[2 3 4 5]	[10 10]	133	237
0.2	[2 3 4 5]	[10 13]	136	234
0.2	[2 3 4 5]	[14 16]	136	231
0.2	[7 8 9]	[10 10]	237	334
0.2	[7 8 9]	[10 13]	255	341
0.2	[7 8 9]	[14 16]	247	342
0.3	[2 3]	[10 10]	95	173
0.3	[2 3]	[10 13]	96	172
0.3	[2 3]	[14 16]	96	173
0.3	[2 3 4 5]	[10 10]	134	235
0.3	[2 3 4 5]	[10 13]	138	233
0.3	[2 3 4 5]	[14 16]	137	229
0.3	[7 8 9]	[10 10]	238	332
0.3	[7 8 9]	[10 13]	255	339
0.3	[7 8 9]	[14 16]	248	340

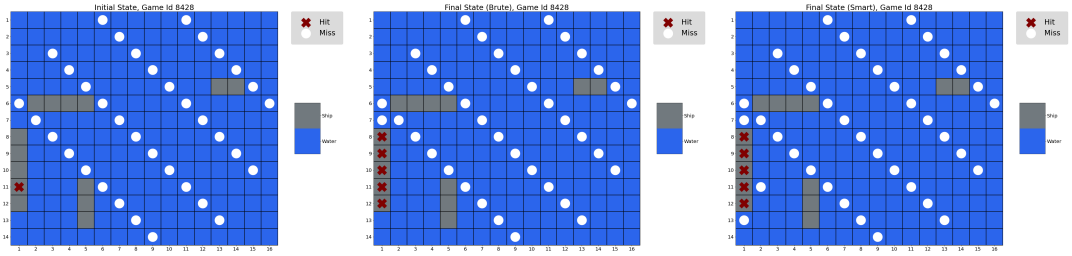

Figure 4: Illustrating the sink ship algorithms. L to R: The start setting, the guesses by Brute, the guesses by smart

100 different randomly simulated initial positions. The integration was done such that we find a first hit and give it to the sinking algorithm to sink the ship. If in the process of sinking that ship it finds other ships then those points are given again to the sink the ship till we run out of such points. Once we are out we go back to finding the first hit. At every step we check if all ships are sunk at which point the game is declared complete. The performance of the bots can be seen in Figure 5.

We can see quite clearly that the PDF Brute and the Diag Brute are the best typically taking around 45 guesses to complete. This is better than the average human which takes on average 49 moves to complete a game. The PDF and Diag Smart are slightly worse typically but with similar upper bounds. The Random Brute and Random Smart algorithms are significantly worse taking typically 60-70 guesses and even going over 90 in some cases.

Note: As a fun experiment we played a few games against our best bot, the PDF-Brute. In 10 games against a human it managed to win 3 and came close in 2 others. This we feel is quite impressive given

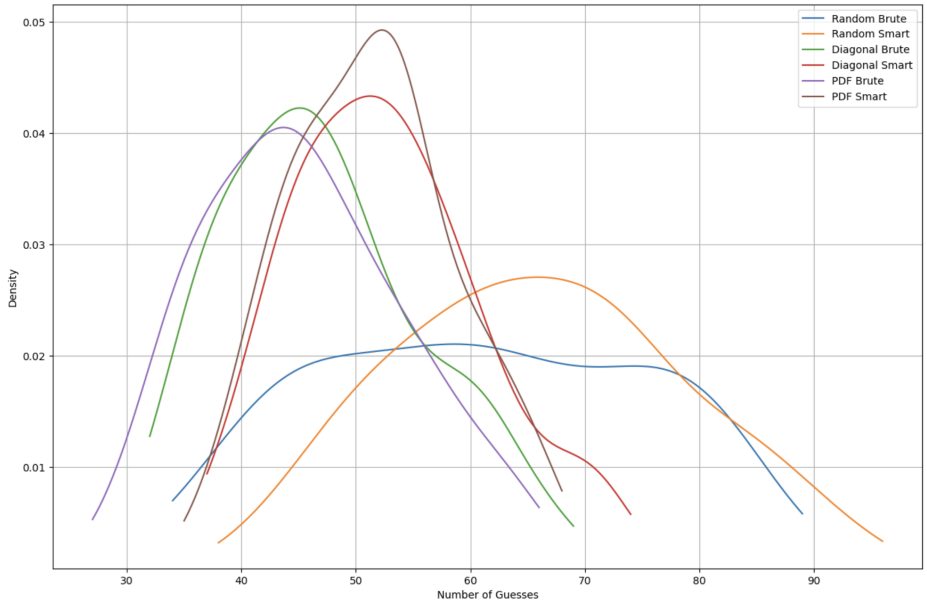

Figure 5: Distribution of number of guesses for different strategies to win

that we are a group of quite decent battleship players.

3.4 Conclusion

Overall, we can say that combining the best of both runtime and average number of guesses needed, while keeping in mind the computation a human can do while playing, our suggestion is to use the Diagonal algorithm. However, as the game progresses a little PDF becomes a better choice often doing better by 1 guess which as we all know can be the different between winning and losing.

Even though the smart sink is named smart it takes more guesses to sink the ship and takes longer run time so the clear choice is the brute force approach. This is not very surprising as sinking the ship once you know the first hit is a relatively straightforward task.

In combination, the smart sinking algorithm did not provide the benefit we were hoping. So the best choice by the numbers it to use the PDF algorithm to find a hit and then sink it using brute force approach. However, as discussed before the PDF algorithm is very cumbersome for a human playing to maintain. So our suggestion is to start out with the diagonal approach to find the first hit and use the brute force to sink ships. Once 20 - 30% of the board has been guessed switch to an approximate PDF calculation to find yourself the first hit. It should be much simpler now. Hope this improves your Battleship skills, we certainly improved ours!