Group 31: Battleship

Edward Chen: 18etc4

Ryan Licandro: 18rl50

Courtney McNamara: 14cpm3

Samantha Stinson: 15ss190

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

December 7, 2020

Abstract

Battleship is a war-themed, well known two player board game where the players try to guess the location of their enemies hidden ships and sink them. To play the game, each player places ships of different sizes on their own board. Neither the player nor their opponent can see the others board. Once setup, the players take turns guessing coordinates on the other player's grid to try to identify a square that contains a ship. The game is won when one player hits all the ship coordinates of the other player, sinking all the enemy ships.

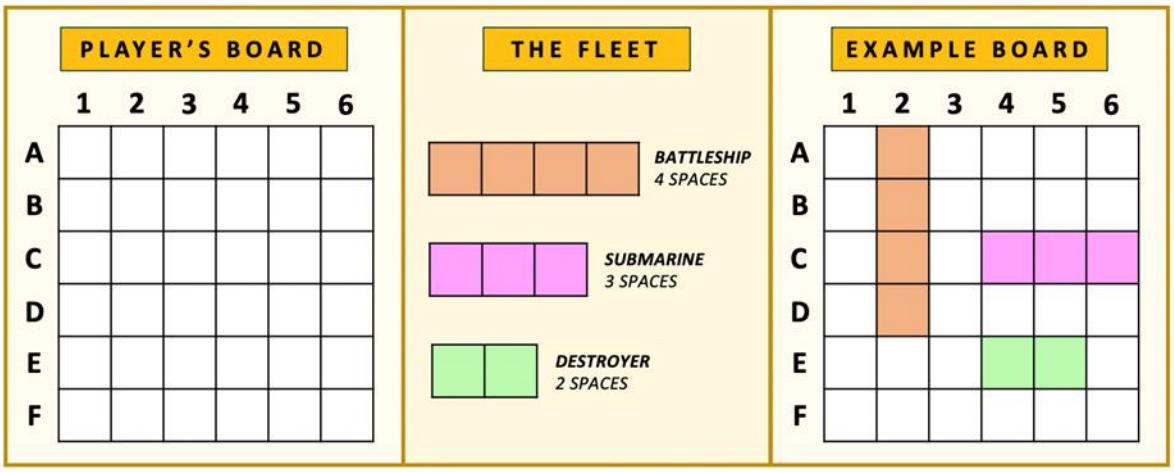


Figure 1: Representation of Battleship Board

The problem is how to model the placement of the ships on a two-dimensional (2D [x, y]) board. To place the ships, the team found that their size, orientation, and permittable position on the grid were essential to the game and became the propositions that were used in the model. When implementing the propositions, the team chose to randomly generate their Boolean value to represent a different game each time. While implementing these propositions, it was necessary to apply constraints that restrict the placement of the starting square of each ship (most top-left square) based on the size and orientation to ensure that the ship remains within the board. The team also constrained the starting square placement of a ship so it would not overlap with that of another ship. While testing the implemented logic, the original board size of a 10 x 10 grid with 5 ships was too large to compute the number of solutions, and the game was reduced to a 6 x 6 board with 3 ships.

Propositions

The propositions of the model are related to both the ships and the grid. The three ships are defined as *s1*, *s2*, and *s3* which refer to the placement of each ship. Each ship has propositions that lie within their properties: size, position, and orientation, which vary game to game.

There are three sizes that a ship may hold. A ship can be 2, 3 or 4 spaces in length (where 2 = "Destroyer", 3 = "Submarine", 4 = "Battleship"), and the sizes are defined as *size2*, *size3*, and *size4* respectively. These propositions are true if they match the size of the ship it is defining. For example, if *s1* occupies 3 spaces, then *size3* would be true.

A ship can take any position that is within the grid so long that it satisfies the constraints of the game. A ships position *Pij* refers to the (*i*, *j*) coordinates of the ship where *i* is the row and *j* is the column. In Figure 1, $PE4 \wedge PE5$ would refer to the position of the Destroyer. Further, a position proposition is true if a ship resides in that location.

Finally, a ship can be placed on the board either vertically or horizontally. This means each ship has its own orientation. An orientation proposition is true if the ship is horizontal and false if it is vertical.

The propositions are summarized below:

s1: true if ship 1 has been placed on the board.

s2: true if ship 2 has been placed on the board.

s3: true if ship 3 has been placed on the board.

Pij: true if a ship is located on coordinate (i, j).

Hij: true if a ship piece has been hit on coordinate (i, j).

horizontal: true if the orientation of a ship is horizontal.

vertical: true if the orientation of a ship is vertical.

size2: true if a ship occupies two squares.

size3: true if a ship occupies three squares.

size4: true if a ship occupies four squares.

So, in Figure 1, *PA1*, *PB4*, and *PC2* are true. In Figure 1, if *s1* is a "Battleship" type, then *s1.size4* and *s1.horizontal* would be true. Whereas if s1 is "Submarine" type, then *s1.size3* and *s1.vertical* would be true.

Constraints

The orientation and position of the ships are randomly generated and placed onto the board, but the ships must adhere to certain constraints to have a valid placement. The size of a ship refers to the number of squares it occupies, and for each ship, there are different restrictions as to which position it may take. For a board that is X by Y, where X is the number of rows and Y the number of columns, the position of a horizontal ship sized n has its most top-left coordinate (the ship's placement coordinate) restricted such that the maximum x-coordinate it can hold is equal to X - (n – 1). Similarly, vertical ships have the maximum y-coordinate as Y - (n – 1). This ensures that the ships stay within the board.

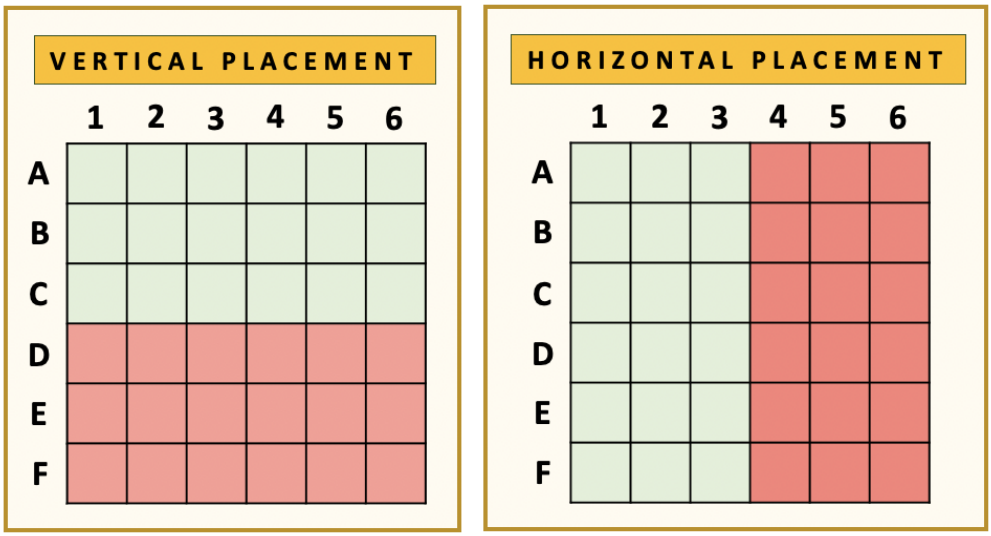


Figure 2: Ship Placement Situation- "If want to place Battleship (size4) onto the board..."

Figure 2 contemplates the placement of a "Battleship" (*size4*) onto the board, where the ship's placement coordinate is restricted depending on its orientation and cannot be placed further down than row C for a vertical ship, or further right that column 3 for a horizontal ship.

The ships must also be placed such that they cannot overlap one another. If a ship is placed at the position *Pij*, then no other ship can hold this position.

The constraints are summarized below:

1. A ship can only be one size:
   $s1.size2 \wedge \neg s1.size3 \wedge \neg s1.size4$
2. A ship can only have one orientation:
   s1.horizontal
3. All the pieces of the ships must remain inside the board:
   In Figure 2, if the goal is to place a horizontal ship of *size4* onto the board, the placement coordinate can be in any of the green squares, but not the red squares.
   $PA1 \vee PA2 \vee PA3 \vee PB1 \vee PB2 \vee PB3 \vee PC1 \vee PC2 \vee PC3 \vee PD1 \vee PD2 \vee PD3 \vee PE1 \vee PE2 \vee PE3 \vee PF1 \vee PF2 \vee PF3 \wedge \neg (PA4 \vee PA5 \vee PA6 \vee PB4 \vee PB5 \vee PB6 \vee PC4 \vee PC5 \vee PC6 \vee PD4 \vee PD5 \vee PD6 \vee PE4 \vee PE5 \vee PE6 \vee PF4 \vee PF5 \vee PF6)$
4. The ships cannot overlap:
   In Figure 1 if the player wants to place another ship, the placement coordinate can be located anywhere on the board that does not already contain a ship piece.
   $PA1 \vee PA3 \vee PA4 \vee PA5 \vee PA6 \vee PB1 \vee PB3 \vee PB4 \vee PB5 \vee PB6 \vee PC1 \vee PC3 \vee PD1 \vee PD3 \vee PD4 \vee PD5 \vee PD6 \vee PE1 \vee PE2 \vee PE3 \vee PE6 \vee PF1 \vee PF2 \vee PF3 \vee PF4 \vee PF5 \vee PF6 \wedge \neg (PA2 \vee PB2 \vee PC2 \vee PC4 \vee PC5 \vee PC6 \vee PD2 \vee PE4 \vee PE5)$

Model Exploration

Phase 1- Requirements and Analysis

In the first phase of model exploration, the team accessed the essential design needs of Battleship game play. It was determined that the most fundamental elements of the game encompass placement, ships and board squares. Research showed that the original Battleship game can vary in grid size, and ships. The game was analyzed to properly generate the models for the necessary application of logic. So, the size of the board was represented by X x Y, such that $X, Y \in \mathbb{N}$. The number of ships was then represented by s, such that $s \in \mathbb{N}$ as the theory needed to be consistent over various board sizes and ship count. The original propositions made in the requirements phase were made with the ideology of finding a model with a winning player. The proposition *xij* was true if a ship was on the location (x,y). The proposition *hxy* was true if the square was hit by the opponent. A player won the game if every *xij* that was true on the opponent's board had a corresponding *hij* that was true. To determine the initial draft of the model's constraints, the team reviewed the official rules of Battleship. It was concluded that all ships can only be placed vertically or horizontally, no ships can overlap, and a ship would only be revealed once the entire ship was hit. These constraints establish the winning condition, which was that a player has won the game once all the opponent's ships have been revealed/ located.

Phase 2- Design and Coding

In the second phase, as the team began to outline design specifications, it became apparent that the model for Battleship was getting very complex. So, the team reduced the model to be more realistic, shifting the focus to a placement model, which if time allotted could be built upon to create the previously mentioned win circumstances.

The team then began implementation of the model in Python with the course project library. The model was initially built upon very redundant code. So, the team investigated a more efficient way of creating the board and ships. The application of the model was fixed using Object-Oriented programming and recursive calls.

With the revised model, the number of solutions grew exponentially as the size of the board and ships increased. At the maximum board size of 10 x10, the computations became too much work for the team's computers due to the vast amount of solutions for the model. To confirm that the model could execute necessary functionalities, the team scaled down the size of the board and number of ships. In these trials, the controlled variable was 3 ships which was tested against different board sizes. The initial feasible board size was a 4 x 4 due to the biggest ship being size of 3. The board size was expanded until unable to compute due to runtime, memory or maximum recursion call errors. After multiple trials, the largest

possible board was a 7x7 board with 3 ships. This reduction allowed the models constraints to be considered and full computation.

At this point, the team was able to receive imperative feedback on the model implementation. From this, the team decided to tweak the placement model on the board. So, size propositions were introduced; size2, size3, and size 4, that if true a ship occupied the corresponding number of squares. As well, the horizontal proposition was established to resolve orientation issues of the ship in predicate logic. The horizontal proposition was true for a horizontally oriented ship and false for a vertically oriented ship.

Table 1: Model Version 1 Outputs

Number of Ships	Board Size (X x Y)	Number of Solutions	Runtime (seconds)
3	6 x 6	6533186235000709060267158057825371437047295487154307196639694971477376	3600

Table 2: Model Version 2 Outputs

Number of Ships	Board Size (X x Y)	Number of Solutions	Runtime (seconds)
3	4 x 4	894,566,400	N/A
3	5 x 5	1,666,984,181,760	5
3	6 x 6	9,690,477,011,927,040	22.7763
3	7 x 7	Chache error: "usedMem all Bytes"	1431

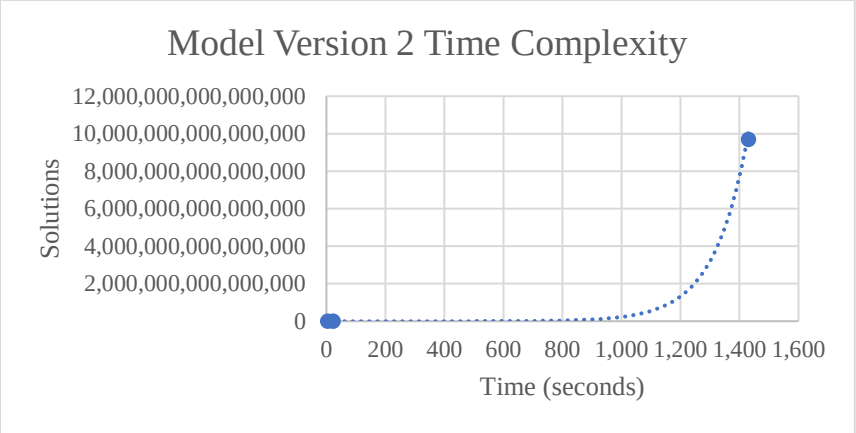


Figure 3: Model Version 2 Graph

Table 3: Model Version 3 Outputs

Number of Ships	Board Size (X x Y)	Number of Solutions	Runtime (seconds)
3	4 x 4	18,432	1.3789
3	5 x 5	144,000	8.8550
3	6 x 6	622,080	17.3471
3	7 x 7	1,975,680	44.9044
3	8 x 8	Recursion Error: "maximum depth exceeded in comparison"	1680

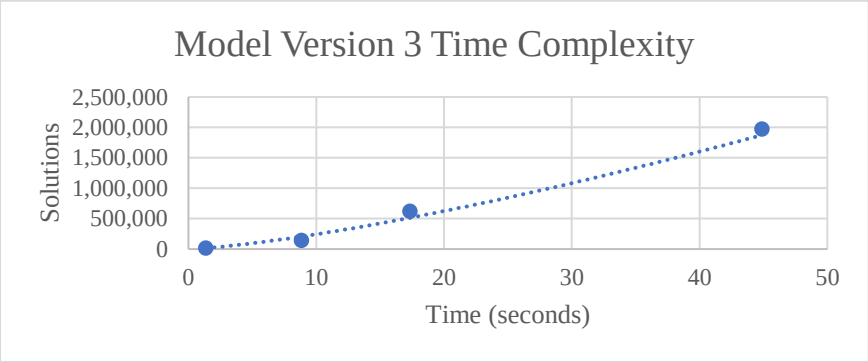


Figure 4: Model Version 3 Graph

Phase 3- Testing and Revising

In the third phase, the team concentrated on creating a fully developed model so all elements of the game could operate. The decided 6 x 6 board, with 3 ships of sizes 2, 3, and 4 was used. To determine if the corresponding ship was placed on the board, the following propositions were added; s1, s2, and s3. Using the new propositions, constraints for size and orientation of all ships could be concluded. These constraints were that a ship could only be one size and a ship can only have one orientation. Another constraint came to light when testing which was that the team had to consider the orientation and size of the ship so that it remains inside the board. From there the possible placement were determined to be all the spaces that were false and not any of the spaces that were true.

The testing phase consisted of eliminating and isolating components of the model to confirm every individual element was fulfilling its requirements. To test, the team created a visual of the output with an ascii table to keep track of the ship's placement. By refocusing the model to functionality rather than additionally computing number of possible solutions the computation became much more feasible.

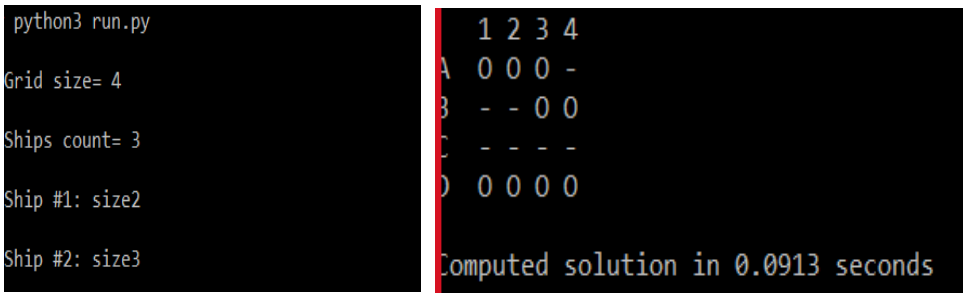


Figure 5: Ascii Table Output Example

After finalizing these important elements, the model's program could run smoothly with advanced propositions and constraints. The following table shows the programs output of ship placement using the constrains and propositions with a reasonable running time.

Table 4: Model Final Version Outputs

Number of Ships	Board Size (X x Y)	Runtime (seconds)
3	4x 4	0.0913
3	5 x 5	0.1763
3	6 x 6	1.23

First-Order Extension

The Battleship model in a predicate logic setting can be extended in the following scenarios.

Single Player

This extension creates a puzzle of a single players placement. One of the main goals of this puzzle is the existence of a unique solution. To create the puzzle, start with a placement solution of the Battleship puzzle and remove redundant information such that the solution of the puzzle remains unique.

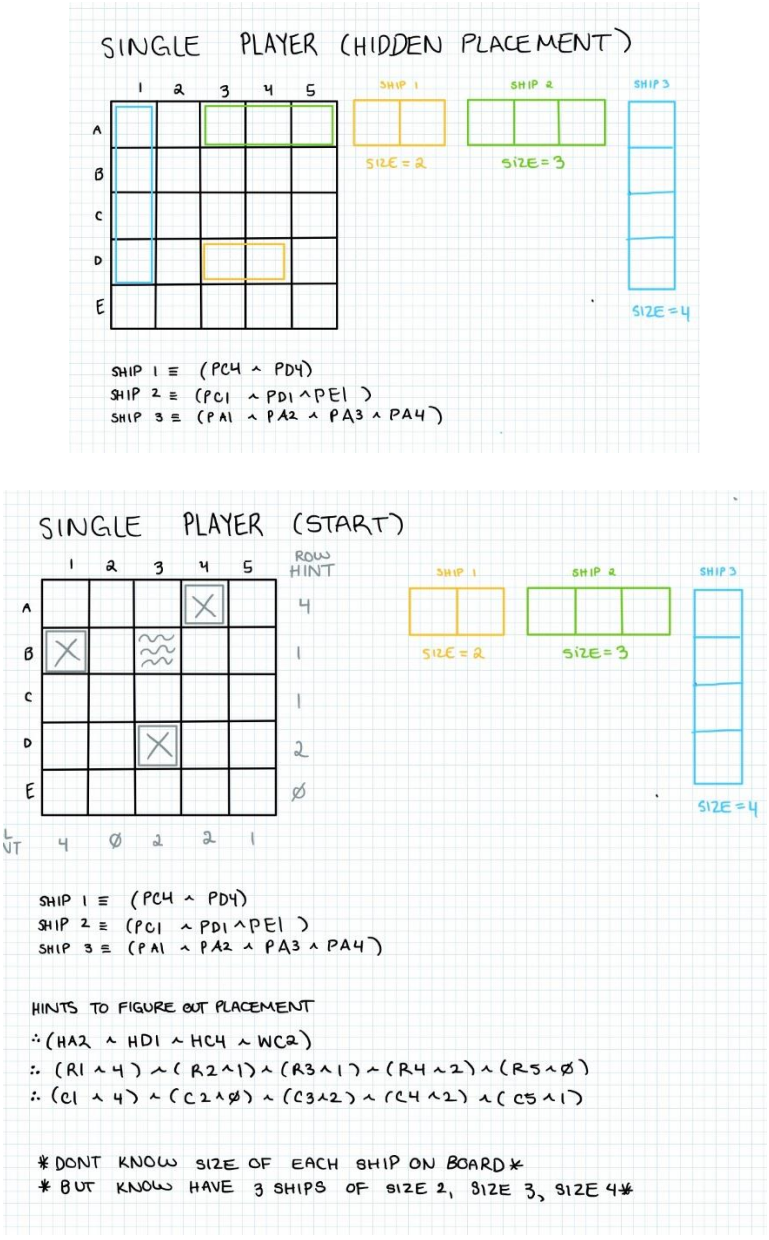


Figure 6: Single Player Extension Visual

In a further extension of this model, there is a strategy presented to finding a solution of the puzzle and maximum/minimum amount of hits available for the player. To solve winning scenarios with a strategy, instead of checking every square independently like in the previous constraints, the new goal is to check every possible winning situation for a horizontally placed ship in a row. Then, assuming a winning row exists, can look for horizontally oriented ship within that row in the scope of all rows on the board. This reasoning can be repeated for columns with vertically placed ships. Then these conditions could reveal near winning answers of opponent's placements of ships.

Rows: "For row i, it is known that for all columns y of placement there exists a solution with a horizontally placed ship, deduced by checking each k row."

Ship s of size=2, $\exists i\, R(i), \forall y(R(y) \rightarrow (\exists k\, P(k,y) \wedge P(k+1,y)) \vdash (P(i,y) \wedge P(i+1,y))$

Using copy of premise 1.1 to create 'actual i' variable, can deduce that for row y, know that if there exists a k of horizontal placed ship, then there is a partial row that exists in the solution to be hit for a ship.

Ship s of size=3, $R(i), \forall y(R(y) \rightarrow (\exists k\, P(k,y) \wedge P(k+1,y) \wedge P(k+2,y)) \vdash (P(i,y) \wedge P(i+1,y) \wedge P(i+2,y))$

Ship s of size=4, $R(i), \forall y(R(y) \rightarrow (\exists k\, P(k,y) \wedge P(k+1,y) \wedge P(k+2,y) \wedge P(k+3,y)) \vdash (P(i,y) \wedge P(i+1,y) \wedge P(i+1,y) \wedge P(i+3,y))$

…

Therefore, for ship s of size=n, $R(i), \forall y(R(y) \rightarrow (\exists k\, P(k,y) \wedge P(k+1,y) \wedge \ldots \wedge P(k+n,y)) \vdash (P(i,y) \wedge P(i+2,y) \wedge \ldots \wedge P(i+n,y))$

Columns: "For column i, it is known that for all rows y of placement there exists a solution with a vertically placed ship, deduced by checking each k column."

Ship s of size=2, $C(j), \forall x(C(x) \rightarrow (\exists k\, P(k,x) \wedge P(k+1,x)) \vdash (P(j,x) \wedge P(j+1,x))$

Using copy of premise 1.1 to create 'actual i' variable, can deduce that for column x, know that if there exists a k of vertical placed ship, then there is a partial row that exists in the solution to be hit for a ship.

Ship s of size=3, $C(j), \forall x(C(x) \rightarrow (\exists k\, P(k,x) \wedge P(k+1,x) \wedge P(k+2,x)) \vdash (P(j,x) \wedge P(j+1,x) \wedge P(j+2,x))$

Ship s of size=4, $C(j), \forall x(R(x) \rightarrow (\exists k\, P(k,x) \wedge P(k+1,x) \wedge P(k+2,x) \wedge P(k+3,x)) \vdash (P(j,x) \wedge P(j+1,x) \wedge P(j+2,x) \wedge P(j+3,x))$

…

Therefore, for ship s of size=n, $C(j), \forall x(R(x) \rightarrow (\exists k\, P(k,x) \wedge P(k+1,x) \wedge \ldots \wedge P(k+n,x)) \vdash (P(j,x) \wedge P(j+1,x) \wedge \ldots \wedge P(j+n,x))$

Strategies

Repeating random attempts is not a good strategy, if player wants to optimize amount of hits needs to win. Studies have shown that this strategy of picking a random square to shoot for each turn simulated over 100 million games require 96 turns to win 50% of games, given that it would not be defeated before that time. This analysis shows that random attempts are very time consuming.

Method 1: Checkerboarding

Create checkerboard on grid, only fire on every other square. Since every ship must have component on both types of squares, by only firing on every other square, half of the locations on are eliminated, which results in a better probability.

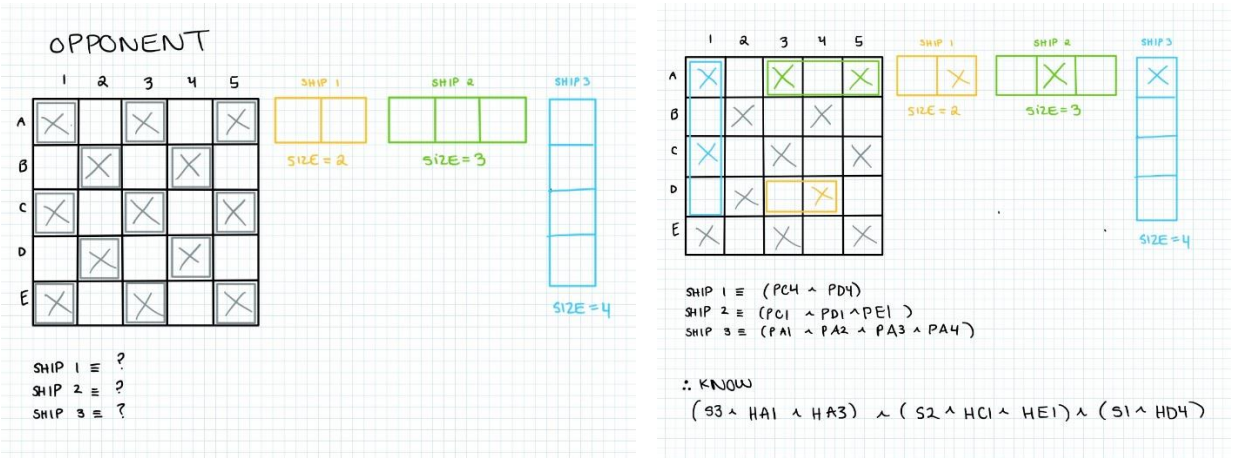


Figure 7: Checkerboarding Visual

Method 2: Hunt and Target

If player hits a ship, they should then open up surrounding squares by hitting adjacent to hit of that ship.

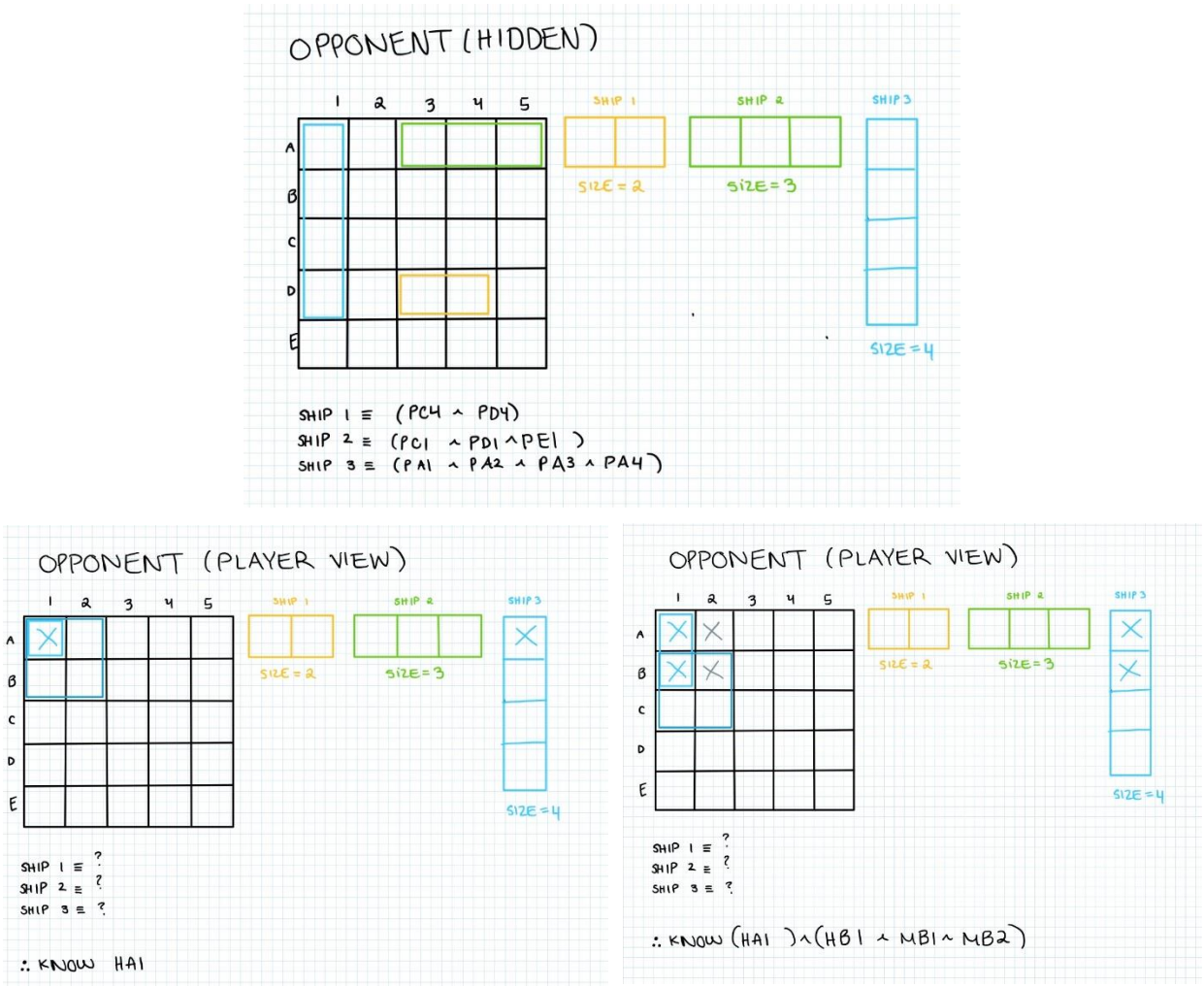


Figure 8: Hunt and Target Visual

Method 3: Probability Density Functions

The goal is to think of the board as plot, some positions have more likeliness than others. A function that describes relative likelihood that a random variable takes on a given value. Redder squares are higher likelihood, since the center is more likely, due to its ability to have more placement options.

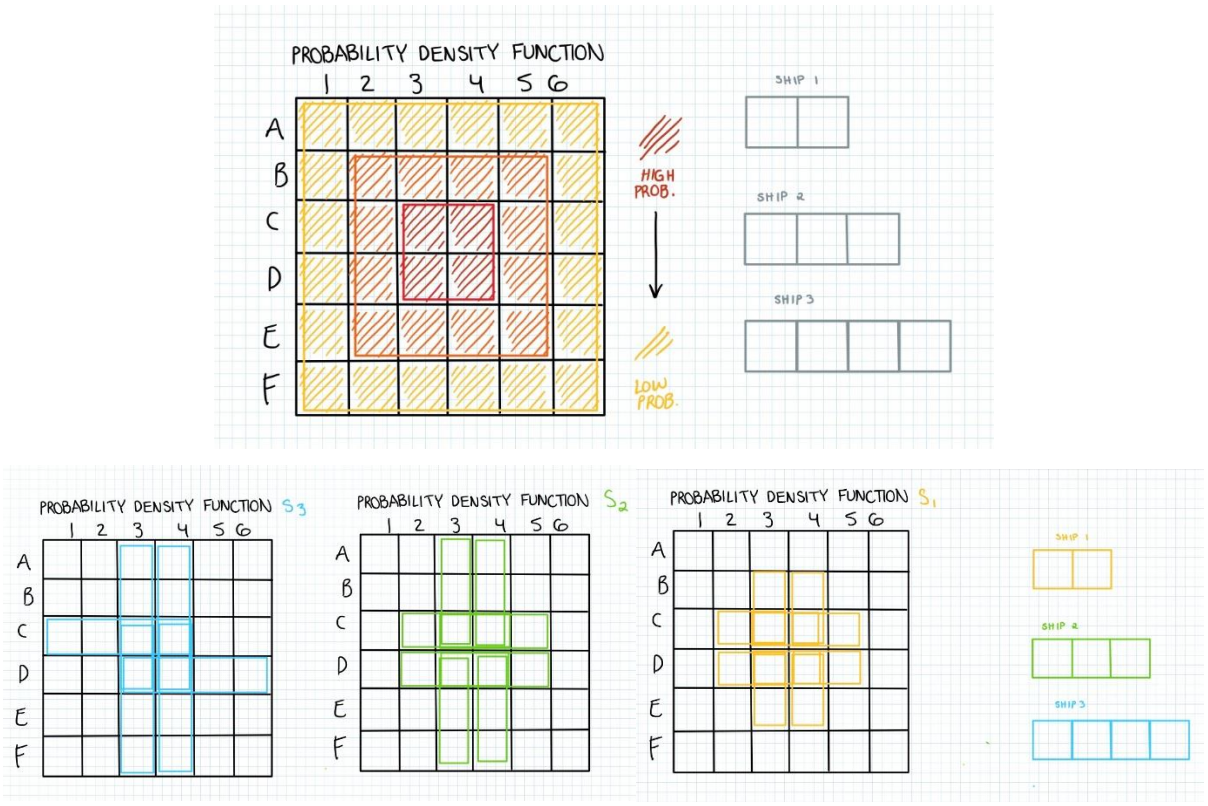


Figure 9: PDF Visual

Multiple Players

Another extension can be to enhance the game to have it be a p player game. The game play will consist of each player taking a turn, where that player will have the advantage of choosing a particular opponent board to hit. For any individual player p if there are multiple players, then p gets to select an opponent from the set of players. “For all players p in a multiple player scenario, then there exists an individual player i, who either i selects an opponent from set of players and aims, or i gets has an opponent try to aim at their ships or i waits.”

Let *M(p)* be multiple players, let *SO(p,i)* be player selects opponent, let *OA(p,i)* be opponent aims at player, let *W(p,i)* be player waits.

$$(\forall p\ M(p)) \rightarrow (\exists i\ SO(p,i) \ \vee OA(p,i) \vee\ W(p,i)\ )$$

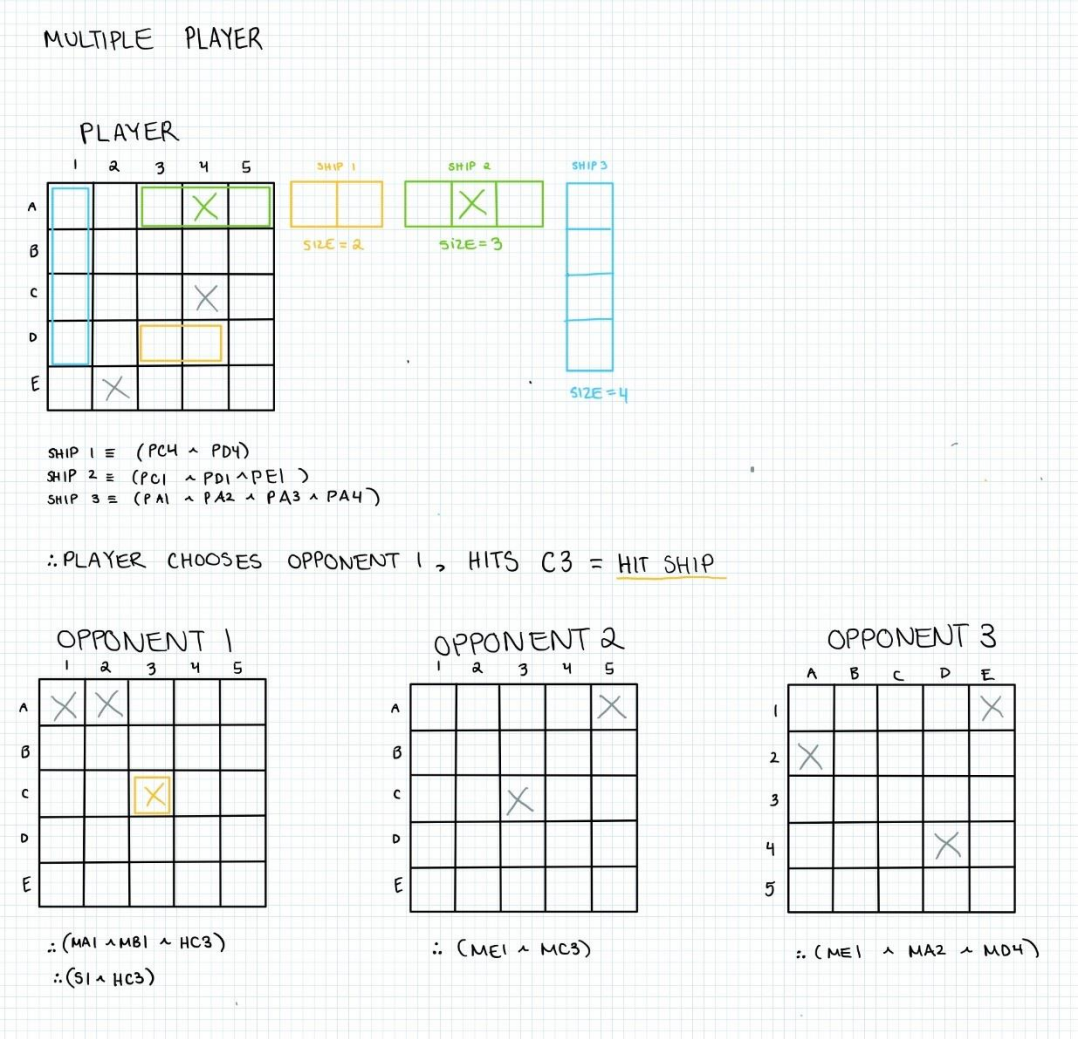


Figure 10: Multiple Players Extension Visual

Added Resources

Bombs

This extension has an additional feature to speed up the gameplay. Firstly, it can include bombs which when hit will reveal information about coordinates around it. “For all squares on x,y coordinates on a grid, there exists an individual bomb b, if hits reveals nearby location or waits until hit.”

Assume *P(x,y)* is object/resource placed on grid including resources and ships. Let *G(x,y)* be square on grid, let *B(b)* be bomb, let *H(x,y)* be hit objects location, *S(s)* be ships, *M(x,y)* be missed hit of any objects.

$(\forall x \forall y\, G(x,y)) \rightarrow ((\exists b\, B(b) \wedge H(x,y)) \rightarrow (\exists s\, G(x+1,y) \wedge H(x+1,y) \wedge P(x+1,y) \wedge S(s)) \vee (G(x+1,y) \wedge M(x+1,y)) \vee (\exists s\, G(x,y+1) \wedge H(x,y+1) \wedge P(x+1,y) \wedge S(s)) \vee (G(x,y+1) \wedge M(x,y+1))$

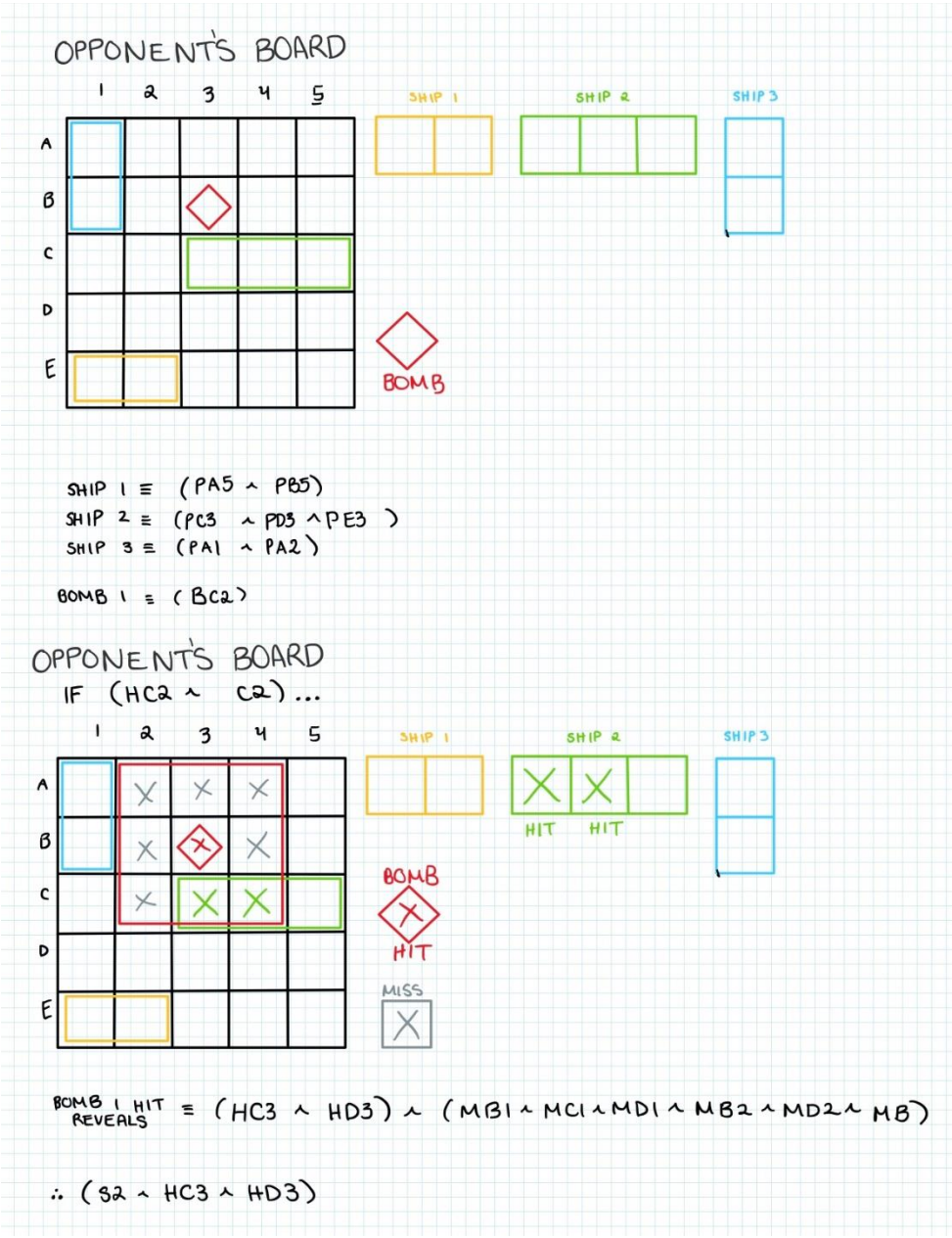


Figure 11: Bombs Resource Visual

Water

This extension could include the water feature, where there must be mandatory spacing around the ships. If know a ship consist of one segment, all neighboring squares must be filled with water thus making

them a miss. “For all squares on x,y coordinates on a grid, there exists a ship s, if hits than know either adjacent horizontal or adjacent vertical square is empty/ is water.”

Let *G(x,y)* be square on grid, let *S(s)* be ship, let *H(x,y)* be hit location, let *M(x,y)* be miss hit of ships, so hit water which is *W(x,y)* for surrounding squares around ship.

$$(\forall x \forall y\, G(x,y)) \rightarrow ((\exists s\, S(s) \wedge H(x,y)) \rightarrow (G(x+1,y) \wedge H(x+1,y) \wedge P(x+1,y) \wedge S(s)) \vee (G(x+1,y) \wedge W(x+1,y) \wedge M(x+1,y)) \vee (G(x,y+1) \wedge H(x,y+1) \wedge S(s)) \vee (G(x,y+1) \wedge M(x,y+1) \wedge W(x+1,y)))$$

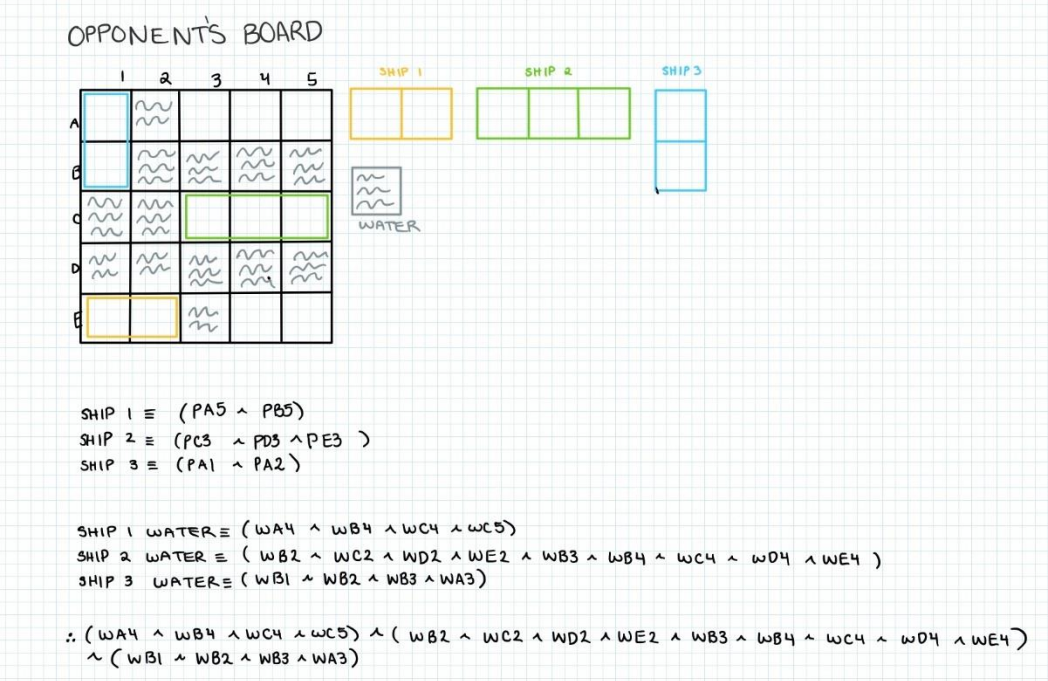


Figure 12: Water Resource Visual

Board Grid Various Sizes

In terms of the board, the grid can be extended to a larger or smaller size for all players. “So, for all squares of grid *G(x,y)*, there exists a square for that size board”.

$$\forall x \in X, \forall y \in Y$$

$$\exists\, G(x,y) = T \vee F$$

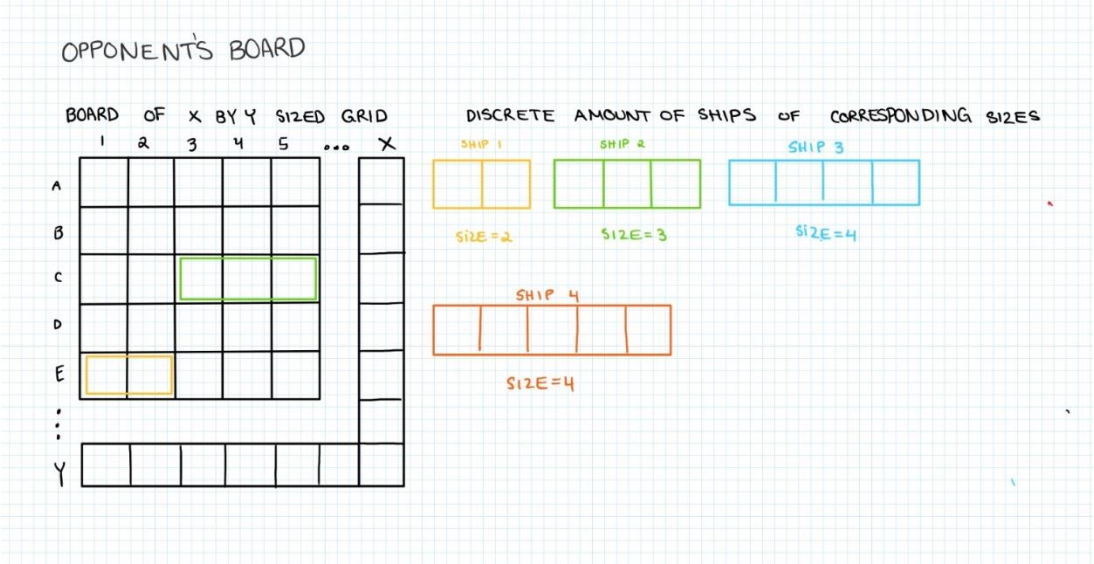


Figure 13: Grid Extension Visual

Ships of Various Amounts

To quantify the amount of ships s with predicates the following statement holds for all ships of specific size. So, for all ships S for a player, a ship placed on the board in either a horizontal or vertical direction.

Let *L(n,s)* be length of ship and assume n is the size of the corresponding ship. Let *S(s)* be ships, let *P(x,y)* be placed on board.

$$\forall s \in S,\ \exists\, S(s) = T \vee F$$

$$\forall x \in X, \forall y \in Y$$

$$\forall s\ (\exists n\, S(s) \wedge L(n,s) \wedge (P(x,y) \wedge P(x+1,y) \wedge \ldots \wedge P(x+n,y))) \vee ((S(s) \wedge (P(x,y) \wedge P(x,y+1) \wedge \ldots \wedge P(x,y+n))$$

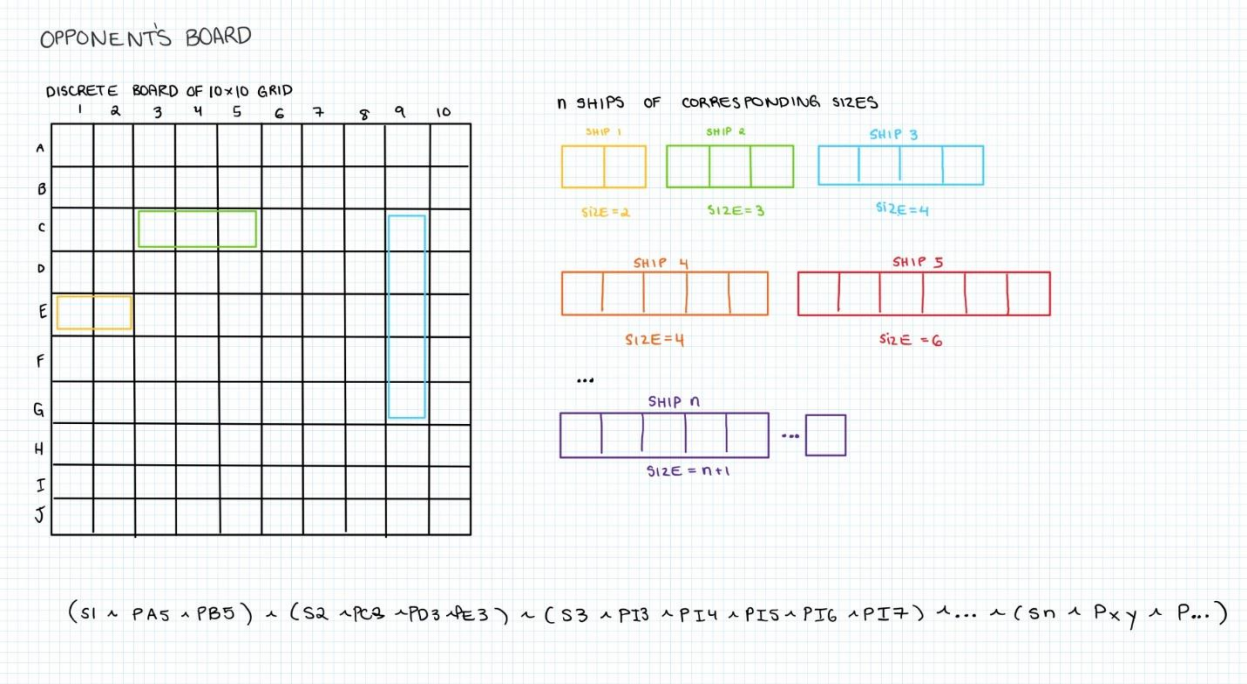


Figure 14: Ships Extension Visual

Jape Proofs

Proof 1

The ships must be placed such that there are no neighboring ships in any of the immediate surrounding squares so that there is no overlap.

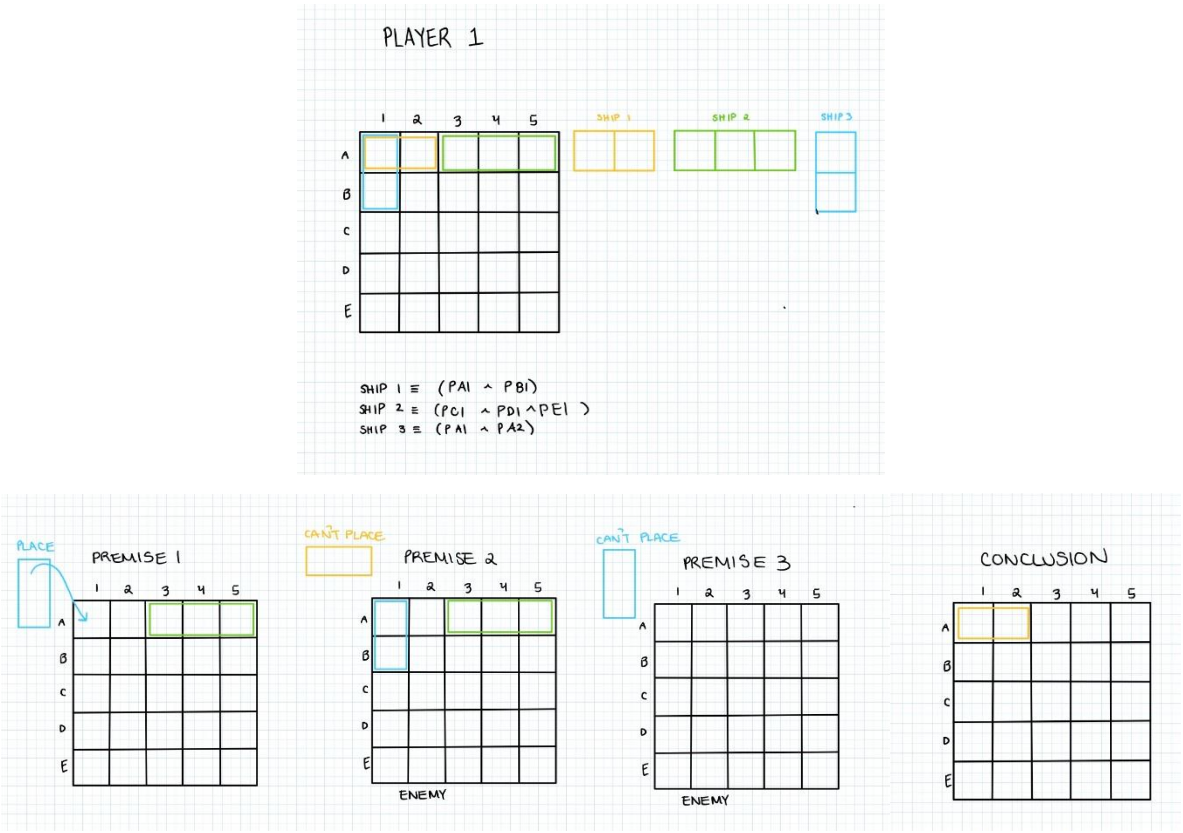


Figure 15: Jape Proof 1 Visual

Let ships of a player be $si \in S$ where ship 1 is *S1*, ship 2 is *S2* where $i \leq 3$. Let *PA1* be top right square of grid has ship placed, let *PB1* be next right square of grid has ship placed, where *Pxy* is $x \in X, y \in Y$ coordinate on grid.

If *S2* is placed on *(C1∧D1∧E1)*, then know can place *S3* on $(A1 \wedge A2)$. Know it is possible to place *S2* and *S3* and not *S1*, which means $(C1 \wedge D1 \wedge E1) \wedge (A1 \wedge A2) \wedge ! (A1 \wedge B1)$. It has been decided to not place *S3* on $(A1 \wedge A2)$. Therefore, it can be deduced that *S1* is placed at $(A1 \wedge B1)$.

1: (S2→S3), (S2∧S3∧¬S1), ¬S3 premises
2: S2∧S3 ∧ elim 1.2
3: S2 ∧ elim 2
4: ¬S1 ∧ elim 1.2
5: ¬S2 → MT 1.1,1.3
6: ⊥ ¬ elim 3,5
7: S1 contra (constructive) 6

1: (PC1∧PD1∧PE1)→(PA1∧PA2) premise
2: (PC1∧PD1∧PE1)∧(PA1∧PA2)∧¬(PA1∧PB1) premise
3: ¬(PA1∧PA2) premise
4: ¬(PC1∧PD1∧PE1) → MT 1,3
5: ¬(PA1∧PB1) ∧ elim 2
6: (PC1∧PD1∧PE1)∧(PA1∧PA2) ∧ elim 2
7: PC1∧PD1∧PE1 ∧ elim 6
8: ⊥ ¬ elim 7,4
9: (PA1∧PB1) contra (constructive) 8

Figure 16: Jape Proof 1

Proof 2

This proof occurs during game play. In this game, each player must place five ships on their board, two of the ships must be size 2, and 2 of the ships must be size 3. The final ship can be either 2 squares or 3 squares. The player has hit and sunk three of the opponent's ships so far: one of size 3 and two of size 2. Based on the current board want to prove that it is know where to target the missiles to sink the last two ships and win the game.

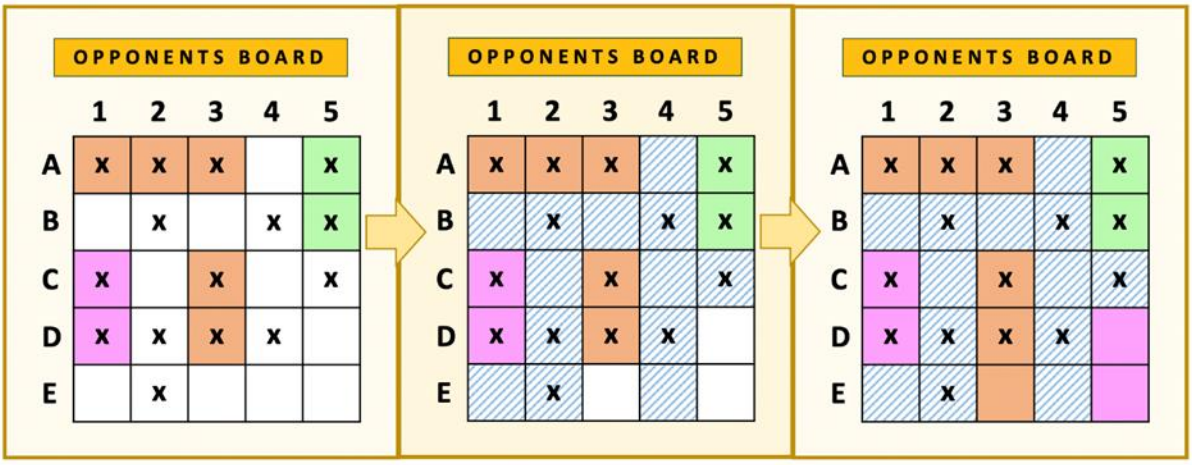


Figure 17: Jape Proof 2 Visual

Based on the ships that have been sunk so far, the remaining two ships must include one of size 3, and that the other ship may be of either size. In order to determine the size of the fourth ship the spacings can be checked around the ships that have been sunk thus far. This reveals that the remaining positions to be hit are located at HC5, HE4, and HE5, and based of this information the player knows where to target in order to win the game.

1.	$S4 \wedge HC3 \wedge HC4, S4 \wedge HC3 \wedge HC4 \rightarrow HC5, S5 \rightarrow HE4 \wedge HE5$	premises
2.	$HC4$	$\wedge$ elim 1.1
3.	$S4 \wedge HC3$	$\wedge$ elim 1.1
4.	$HC3$	$\wedge$ elim 3
5.	$HC3 \wedge HC4$	$\wedge$ intro 4, 2
6.	$HC5$	$\rightarrow$ elim 1.2, 1.1
7.	$HC3 \wedge HC4 \wedge HC5$	$\wedge$ intro 5, 6
8.	$S4$	assumption
9.	$HC3 \wedge HC4 \wedge HC5$	hyp 7
10.	$S4 \rightarrow HC3 \wedge HC4 \wedge HC5$	$\rightarrow$ intro 8-9
11.	$(S4 \rightarrow HC3 \wedge HC4 \wedge HC5) \wedge (S5 \rightarrow HE4 \wedge HE5)$	$\wedge$ intro 10, 1.3

Figure 18: Jape Proof 2

Proof 3

This proof occurs during game play. It is known that the opponent has placed three ships on their board, one of size three squares and two of size two squares. The player has hit and sunk two of the ships so far

and are looking to sink the final ship. The two ships that are sunk are of size three and size two, so the player must know the final ship must be two squares. So, the player has hit one of the squares on the final ship and must hit the remaining square to win the game.

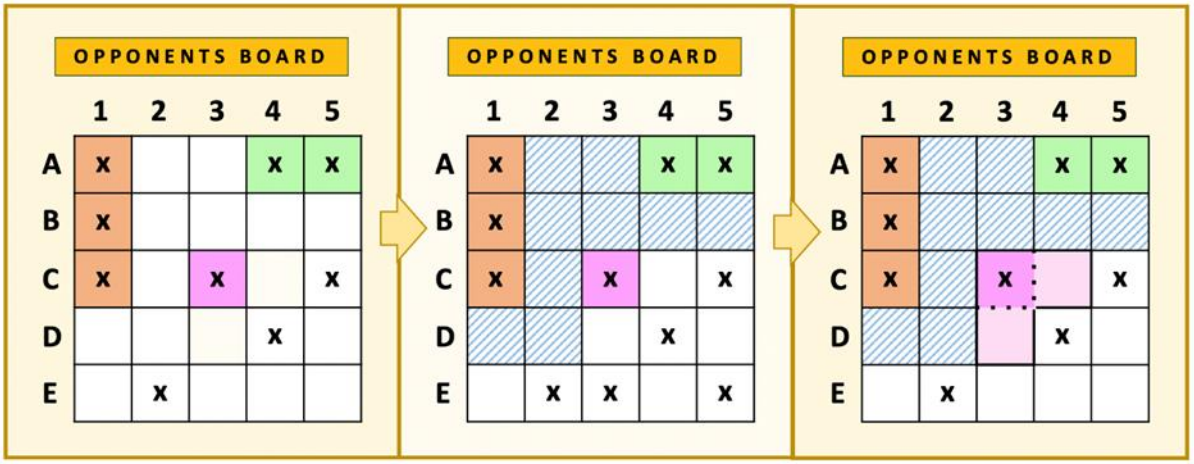


Figure 19: Jape Proof 3 Visual

Based on the ships' locations and the constraints of the game can be observed that the remaining square to be hit must be at HD3 or HC4.

1.	$S2 \wedge HC3 \rightarrow (HC3 \wedge HC4 \wedge \neg HD3) \vee (HC3 \wedge HD3 \wedge \neg HC4)$	premises
2.	$\neg HC4 \rightarrow HD3, \neg HD3 \rightarrow HC4, HC3, S2$	premises
3.	$S2 \wedge HC3$	$\wedge$ intro 2.4, 2.3
4.	$(HC3 \wedge HC4 \wedge \neg HD3) \vee (HC3 \wedge HD3 \wedge \neg HC4)$	$\rightarrow$ elim 1, 3
5.	$HC3 \wedge HC4 \wedge \neg HD3$	assumption
6.	$\neg HD3$	$\wedge$ elim 5
7.	$HC4$	$\rightarrow$ elim 2.2, 6
8.	$HC4 \wedge \neg HD3$	$\wedge$ intro 7, 6
9.	$S2 \wedge HC3 \wedge (HC4 \wedge \neg HD3)$	$\wedge$ intro 3, 8
10.	$S2 \wedge HC3 \wedge (HC4 \wedge \neg HD3) \vee (HD3 \wedge \neg HC4)$	$\vee$ intro 9
11.	$HC3 \wedge HD3 \wedge \neg HC4$	assumption
12.	$\neg HC4$	$\wedge$ elim 11
13.	$HC3 \wedge HD3$	$\wedge$ elim 11
14.	$HD3$	$\wedge$ elim 13
15.	$HD3 \wedge \neg HC4$	$\wedge$ intro 14, 12
16.	$S2 \wedge HC3 \wedge (HC4 \wedge \neg HD3) \vee (HD3 \wedge \neg HC4)$	$\vee$ intro 15
17.	$(S2 \wedge HC3 \wedge (HC4 \wedge \neg HD3) \vee (HD3 \wedge \neg HC4))$	$\vee$ elim 4, 5-10, 11-16

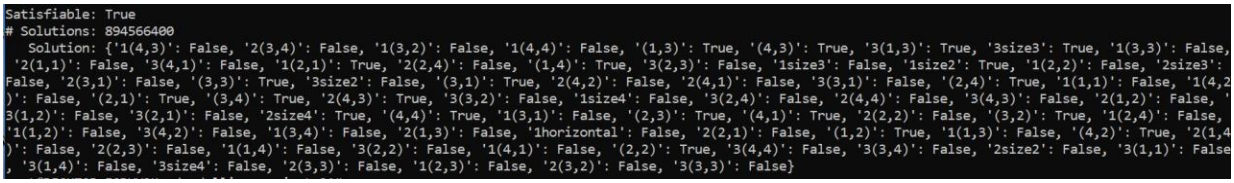
Figure 20: Jape Proof 3

Appendix

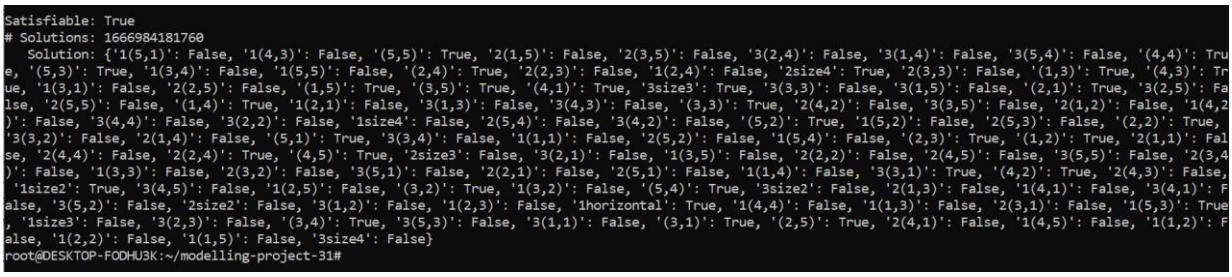
Phase 1

Phase 2 a

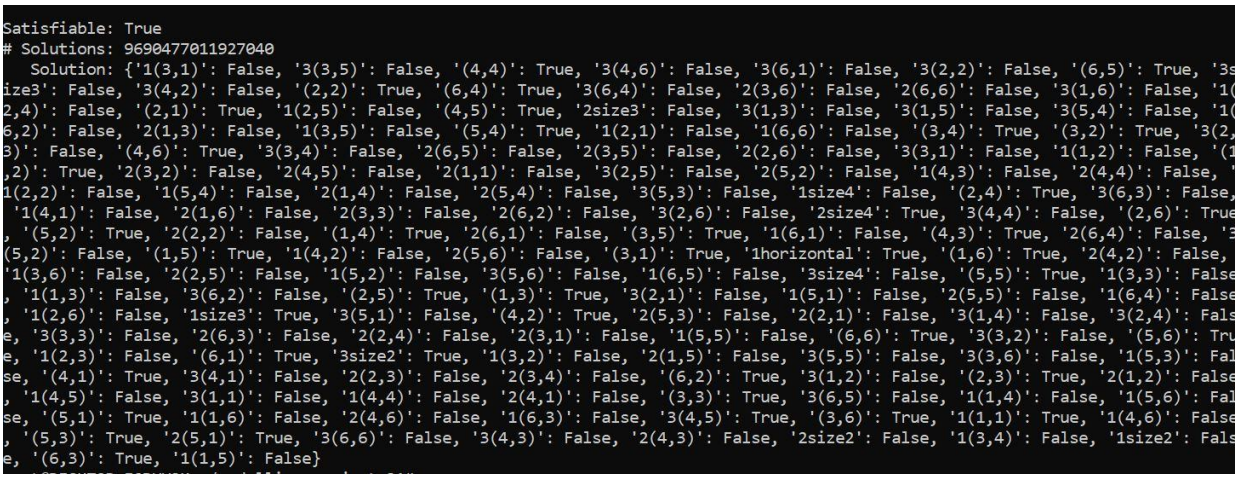
4 x 4 – 3 Ships



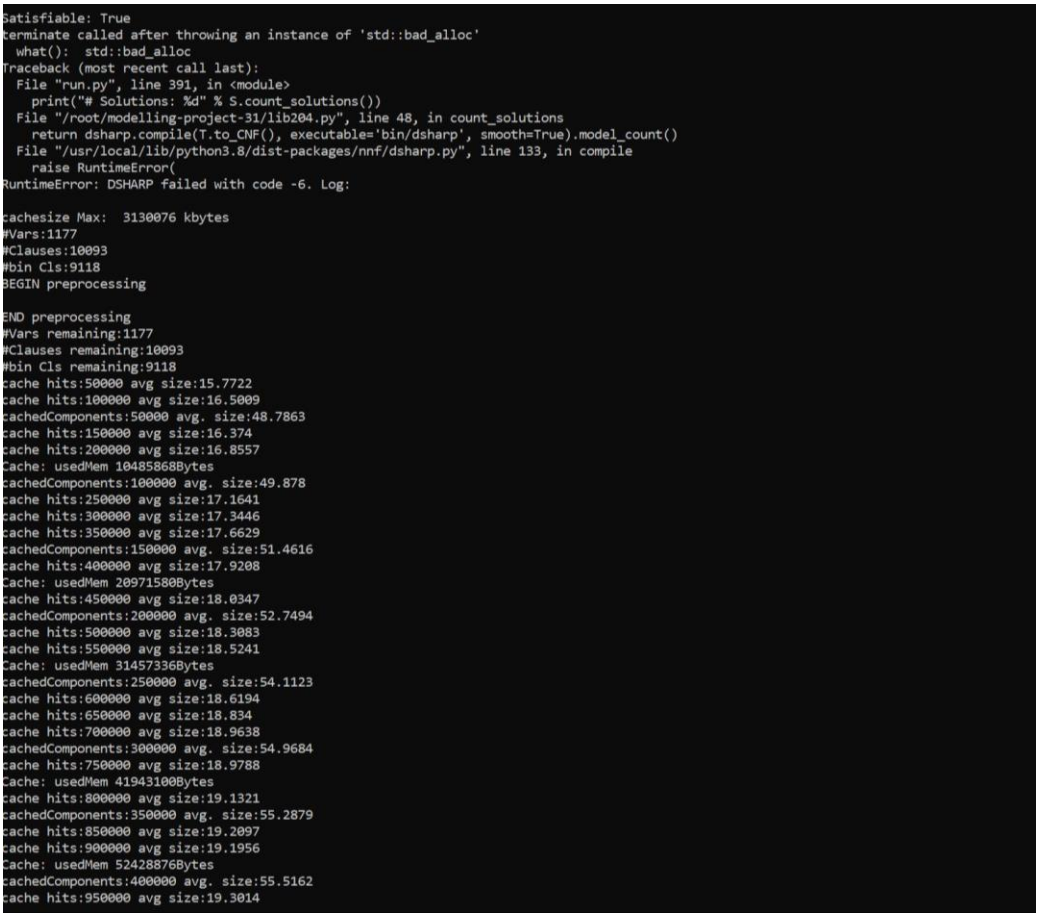
5 x 5 – 3 Ships



6 x 6 – 3 Ships



7 x 7 – 3 Ships



…

```
Cache: usedMem 440402040Bytes
cache hits:7350000 avg size:21.0586
cache hits:7400000 avg size:21.0765
cachedComponents:3100000 avg. size:60.9756
cache hits:7450000 avg size:21.0805
Cache: usedMem 450887756Bytes
cache hits:7500000 avg size:21.0892
cache hits:7550000 avg size:21.0979
cachedComponents:3150000 avg. size:61.0137
cache hits:7600000 avg size:21.1003
cache hits:7650000 avg size:21.1127
Cache: usedMem 461373452Bytes
cachedComponents:3200000 avg. size:61.0455
cache hits:7700000 avg size:21.1107
cache hits:7750000 avg size:21.1114
cachedComponents:3250000 avg. size:61.0063
cache hits:7800000 avg size:21.1039
Cache: usedMem 471859240Bytes
cache hits:7850000 avg size:21.0932
cache hits:7900000 avg size:21.0972
cachedComponents:3300000 avg. size:60.9405
cache hits:7950000 avg size:21.1021
cache hits:8000000 avg size:21.1189
Cache: usedMem 482345056Bytes
cachedComponents:3350000 avg. size:60.9538
cache hits:8050000 avg size:21.1313
cache hits:8100000 avg size:21.1308
cachedComponents:3400000 avg. size:60.9614
cache hits:8150000 avg size:21.1506
Cache: usedMem 492830836Bytes
cache hits:8200000 avg size:21.1664
```

Phase 2 b

4 x 4 – 3 Ships

```
Grid size= 4

Ships count= 3

Ship #1: size2

Ship #2: size3

Ship #3: size4

Satisfiable: True
# Solutions: 18432
   Solution: {'2size3': False, '2size2': True, '2s(3,3)': False, '3s(2,3)': False, '2size4': False, '2s(2,3)': False, '2s(2,4)': False, '1s(4,4)': False, '3s
ize3': True, '3s(1,2)': False, '3s(4,4)': False, '3s(2,4)': True, '3s(4,2)': False, '3s(4,3)': False, '1s(1,4)': False, '1size2': False, '3s(1,3)': False, '3
size2': False, '3s(1,4)': False, '3s(3,2)': False, '1s(1,3)': True, '1s(3,2)': False, '1s(2,2)': False, '2s(2,2)': False, '1s(1,1)': False, '1s(3,4)': False,
 '3vertical': False, '2s(4,1)': False, '2s(4,3)': False, '1vertical': False, '3s(1,1)': False, '3s(3,3)': False, '1size4': True, '2s(4,4)': False, '2s(1,4)':
 False, '1s(4,1)': False, '2s(3,2)': False, '1s(2,1)': False, '3size4': False, '1s(2,3)': False, '2horizontal': False, '2s(3,1)': False, '3s(2,1)': False, '1
s(3,1)': False, '3s(3,1)': False, '2s(2,1)': False, '3s(4,1)': False, '1horizontal': True, '2s(3,4)': False, '1s(1,2)': False, '3s(3,4)': False, '1s(4,3)': F
alse, '1size3': False, '2s(4,2)': False, '3horizontal': True, '2s(1,1)': True, '3s(2,2)': False, '1s(2,4)': False, '1s(4,2)': False, '1s(3,3)': False, '2vert
ical': True, '2s(1,3)': False, '2s(1,2)': False}

Computed solution in 1.3789 seconds
```

5 x 5 – 3 Ships

```
 Grid size= 5

 Ships count= 3

 Ship #1: size2

 Ship #2: size3

 Ship #3: size4

Satisfiable: True
# Solutions: 144000
   Solution: {'3s(4,2)': False, '1s(1,5)': False, '2s(4,3)': False, '2s(2,1)': True, '3s(4,5)': False, '1s(3,1)': False, '1s(2,3)': False, '2s(4,1)': False,
'3s(4,3)': False, '2s(2,3)': False, '1s(2,5)': False, '2s(1,5)': False, '3s(2,1)': False, '1s(5,1)': False, '2size2': False, '2s(5,2)': False, '3horizontal':
 False, '3s(1,5)': False, '3s(3,2)': False, '1s(1,4)': False, '2s(1,4)': False, '2s(3,4)': False, '1s(2,2)': False, '2horizontal': True, '3s(5,3)': False, '2
s(2,2)': False, '1horizontal': False, '2size4': True, '3s(1,1)': False, '3s(4,1)': False, '3s(5,4)': False, '1s(1,3)': True, '1vertical': True, '1s(4,3)': Fa
lse, '1s(4,2)': False, '1s(5,5)': False, '1s(3,3)': False, '1s(2,4)': False, '1s(2,1)': False, '3s(5,1)': False, '1s(3,2)': False, '2s(2,4)': False, '1size4'
: False, '1s(3,5)': False, '2s(3,5)': False, '2s(5,1)': False, '3size4': False, '2vertical': False, '2s(1,2)': False, '2s(3,2)': False, '2s(3,3)': False, '3s
(3,3)': False, '3size2': True, '1s(4,1)': False, '3s(1,4)': False, '2s(1,3)': False, '1s(1,2)': False, '3s(5,5)': False, '1s(3,4)': False, '1s(4,5)': False,
'3s(3,5)': False, '3s(4,4)': False, '2s(5,4)': False, '2s(3,1)': False, '1size3': True, '3s(2,2)': False, '3s(2,3)': False, '2size3': False, '3s(3,4)': False
, '2s(2,5)': False, '3vertical': True, '3s(3,1)': True, '3size3': False, '2s(4,5)': False, '3s(2,4)': False, '1s(5,4)': False, '3s(5,2)': False, '3s(1,2)': F
alse, '2s(5,5)': False, '1s(1,1)': False, '3s(1,3)': False, '2s(4,4)': False, '2s(4,2)': False, '3s(2,5)': False, '1s(4,4)': False, '1s(5,2)': False, '1s(5,3
)': False, '2s(1,1)': False, '2s(5,3)': False, '1size2': False}

Computed solution in 8.8550 seconds
```

6 x 6 – 3 Ships

```
 Grid size= 6

 Ships count= 3

 Ship #1: size2

 Ship #2: size3

 Ship #3: size4

Satisfiable: True
# Solutions: 622080
   Solution: {'1s(4,2)': False, '2s(5,4)': False, '2s(4,1)': False, '2s(6,1)': False, '3horizontal': True, '2s(2,6)': True, '1s(2,4)': False, '3s(4,4)': Fals
e, '2vertical': False, '3s(4,5)': False, '3s(3,5)': False, '3s(4,2)': False, '3s(4,3)': False, '1s(4,6)': False, '1horizontal': True, '3s(1,6)': False, '1s(6
,2)': False, '3s(6,4)': False, '1s(2,5)': True, '2s(2,2)': False, '2s(2,4)': False, '1s(3,2)': False, '1s(1,6)': False, '3size4': False, '2s(1,5)': False, '3
s(3,3)': False, '1s(4,1)': False, '2s(4,4)': False, '2s(4,5)': False, '3s(4,1)': False, '3s(3,6)': False, '1size3': True, '1s(3,4)': False, '1vertical': Fals
e, '2s(6,4)': False, '2s(1,1)': False, '3s(2,5)': True, '3s(2,2)': False, '2size3': False, '1size4': False, '3s(1,5)': False, '1s(1,2)': False, '1s(2,1)': Fa
lse, '2s(4,2)': False, '2s(1,3)': False, '1s(1,5)': False, '2s(6,2)': False, '1s(6,6)': False, '3s(4,6)': False, '3s(1,2)': False, '2s(3,2)': False, '1size2'
: False, '2s(2,5)': False, '1s(6,5)': False, '2s(3,1)': False, '1s(5,6)': False, '2s(2,3)': False, '1s(5,4)': False, '3s(5,5)': False, '3s(2,4)': False, '2s(
1,4)': False, '1s(2,2)': False, '2horizontal': True, '3s(3,2)': False, '2s(5,6)': False, '1s(2,3)': False, '3s(1,1)': False, '1s(5,5)': False, '3vertical': F
alse, '1s(4,4)': False, '3s(3,4)': False, '2s(5,2)': False, '1s(4,3)': False, '2size4': True, '3s(5,6)': False, '2s(6,5)': False, '2s(3,3)': False, '1s(5,3)'
: False, '2s(3,4)': False, '3s(6,2)': False, '3s(6,6)': False, '3s(6,1)': False, '3s(1,4)': False, '2s(4,3)': False, '3s(5,1)': False, '1s(3,1)': False, '3s(
3,1)': False, '1s(1,4)': False, '3s(2,3)': False, '1s(4,5)': False, '3s(1,3)': False, '2s(1,2)': False, '2s(5,1)': False, '1s(3,3)': False, '3size2': True, '
1s(6,1)': False, '3size3': False, '1s(3,5)': False, '1s(2,6)': False, '3s(2,1)': False, '2s(5,5)': False, '2s(6,3)': False, '3s(6,5)': False, '3s(6,3)': Fals
e, '2s(5,3)': False, '2s(6,6)': False, '1s(5,1)': False, '2s(4,6)': False, '3s(5,3)': False, '1s(3,6)': False, '1s(6,3)': False, '1s(5,2)': False, '2s(2,1)':
 False, '3s(5,2)': False, '2s(3,5)': False, '2s(3,6)': False, '2size2': False, '3s(2,6)': False, '3s(5,4)': False, '1s(1,3)': False, '1s(6,4)': False, '2s(1,
6)': False, '1s(1,1)': False}

Computed solution in 17.3471 seconds
```

7 x 7 – 3 Ships

```
Grid size= 7

Ships count= 3

Ship #1: size2

Ship #2: size3

Ship #3: size4

Satisfiable: True
# Solutions: 1975680
```

```
   Solution: {'1s(3,5)': False, '3size2': False, '2s(2,6)': False, '1s(7,3)': False, '3s(2,4)': False, '1s(2,2)': False, '2horizontal': False, '2size4': Fals
e, '3s(5,7)': False, '2s(1,4)': False, '1s(1,3)': False, '2s(4,4)': False, '2s(7,7)': False, '2s(6,6)': False, '2s(7,3)': False, '3s(5,5)': False, '3s(1,4)':
 False, '2vertical': True, '3s(5,3)': False, '3size4': True, '1s(2,6)': False, '2s(2,4)': False, '1s(6,7)': False, '1size2': True, '1s(6,3)': False, '1s(3,3)
': False, '1s(7,5)': False, '1s(7,6)': False, '2s(4,6)': False, '1s(7,1)': False, '3s(4,6)': False, '3vertical': True, '3s(4,4)': False, '2s(7,1)': False, '2
s(1,2)': False, '1s(2,5)': False, '2s(4,2)': False, '3s(6,1)': False, '3s(2,1)': False, '3s(7,6)': False, '2s(1,7)': False, '3s(4,5)': False, '2s(5,6)': Fals
e, '3s(7,5)': False, '3s(1,3)': False, '1s(5,6)': False, '2s(4,5)': True, '2s(6,1)': False, '1s(1,6)': True, '1s(4,2)': False, '1s(4,7)': False, '2s(3,4)': F
alse, '2s(5,2)': False, '2s(7,4)': False, '1s(1,1)': False, '2s(3,2)': False, '1s(3,6)': False, '3s(7,2)': False, '2s(6,2)': False, '2s(2,1)': False, '3s(7,3
)': False, '3s(7,1)': False, '3s(5,6)': False, '3s(2,6)': False, '3s(6,5)': False, '2s(5,5)': False, '2s(2,2)': False, '3s(4,3)': False, '2s(2,7)': False, '1
s(4,3)': False, '3horizontal': False, '2s(2,3)': False, '2s(5,7)': False, '3s(3,1)': False, '3s(4,2)': False, '1s(6,4)': False, '3s(3,2)': False, '3s(6,6)':
False, '3s(2,7)': False, '2s(5,3)': False, '3s(4,1)': True, '1s(1,2)': False, '3s(1,6)': False, '1s(3,7)': False, '2s(4,3)': False, '1s(7,4)': False, '1s(5,5
)': False, '3s(5,1)': False, '1s(5,3)': False, '2s(1,6)': False, '2s(1,1)': False, '2s(7,6)': False, '2s(3,6)': False, '3s(3,3)': False, '2s(2,5)': False, '1
size3': False, '2s(6,4)': False, '3s(2,5)': False, '2s(4,7)': False, '3s(6,4)': False, '1vertical': False, '3s(3,5)': False, '2size3': True, '2s(7,5)': False
, '1s(7,2)': False, '2s(7,2)': False, '2s(3,3)': False, '3s(6,7)': False, '1s(3,4)': False, '3s(1,1)': False, '2s(3,1)': False, '1s(1,5)': False, '1s(2,3)':
False, '1s(5,1)': False, '3s(1,5)': False, '1s(4,6)': False, '1s(6,5)': False, '1s(5,2)': False, '1s(1,4)': False, '1s(4,4)': False, '1s(2,1)': False, '1size
4': False, '1s(5,7)': False, '1s(4,1)': False, '3s(3,6)': False, '3s(5,2)': False, '1s(6,6)': False, '1s(3,2)': False, '3s(7,7)': False, '1s(2,7)': False, '3
s(3,4)': False, '1s(2,4)': False, '2s(4,1)': False, '3s(1,2)': False, '1s(7,7)': False, '3size3': False, '1s(1,7)': False, '3s(4,7)': False, '3s(6,2)': False
, '1s(6,2)': False, '2s(5,1)': False, '3s(2,3)': False, '3s(1,7)': False, '3s(5,4)': False, '2s(3,7)': False, '2s(1,5)': False, '2s(6,5)': False, '1s(4,5)':
False, '2s(6,3)': False, '2s(1,3)': False, '2s(6,7)': False, '1s(5,4)': False, '3s(2,2)': False, '1horizontal': True, '2s(3,5)': False, '2s(5,4)': False, '3s
(7,4)': False, '3s(6,3)': False, '2size2': False, '3s(3,7)': False, '1s(6,1)': False, '1s(3,1)': False}
```

```
Computed solution in 44.9044 seconds
```

8 x 8 – 3 Ships

```
Grid size= 8

Ships count= 3

Ship #1: size2

Ship #2: size3

Ship #3: size4

Satisfiable: True
Traceback (most recent call last):
  File "/usr/local/lib/python3.8/dist-packages/nnf/util.py", line 73, in wrapped
    return memo[self]
  File "/usr/lib/python3.8/weakref.py", line 383, in __getitem__
    return self.data[ref(key)]
KeyError: <weakref at 0x7f4fc9279310; to 'And' at 0x7f4fc13c7130>

During handling of the above exception, another exception occurred:

Traceback (most recent call last):
  File "run.py", line 294, in <module>
    print("# Solutions: %d" % F.count_solutions())
  File "/root/modelling-project-31/lib204.py", line 48, in count_solutions
    return dsharp.compile(T.to_CNF(), executable='bin/dsharp', smooth=True).model_count()
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 493, in model_count
    if decomposable and deterministic and not sentence.smooth():
  File "/usr/local/lib/python3.8/dist-packages/nnf/util.py", line 75, in wrapped
    ret = func(self)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 256, in smooth
    expected = vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
```

…

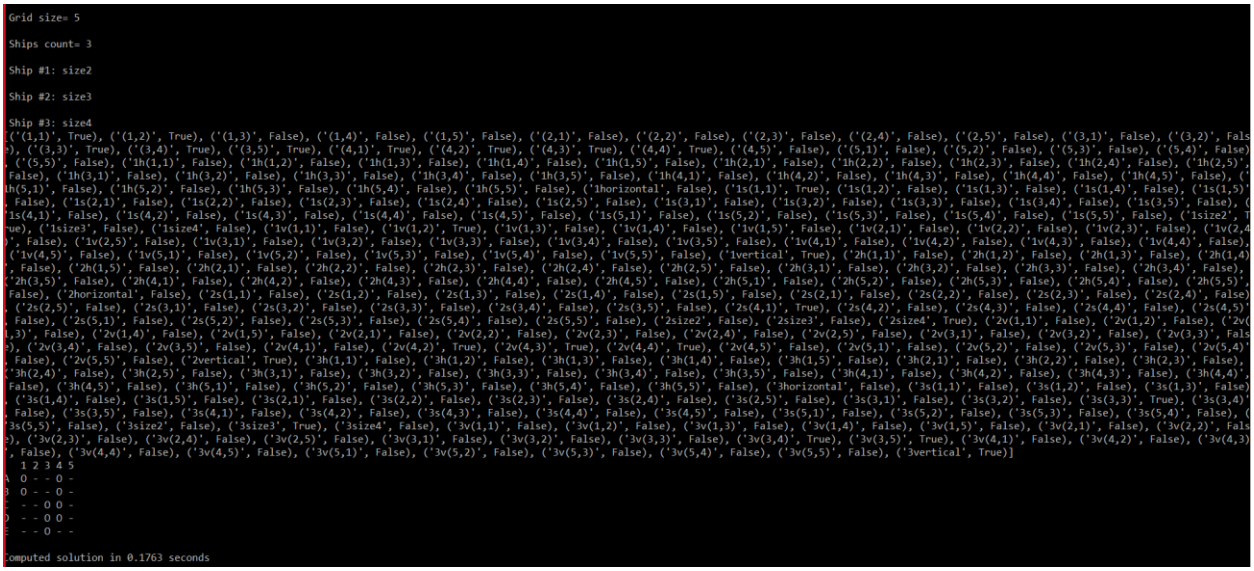
```
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 187, in vars_
    return frozenset(
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 188, in <genexpr>
    v for child in node.children for v in vars_(child)
  File "/usr/local/lib/python3.8/dist-packages/nnf/__init__.py", line 184, in vars_
    if isinstance(node, Var):
  File "/usr/lib/python3.8/abc.py", line 98, in __instancecheck__
    return _abc_instancecheck(cls, instance)
RecursionError: maximum recursion depth exceeded in comparison
```

Phase 3

4 x 4 – 3 Ships

```
Computed solution in 0.0932 seconds
# python3 run.py

 Grid size= 4

 Ships count= 3

 Ship #1: size2

 Ship #2: size3

 Ship #3: size4
[('(1,1)', True), ('(1,2)', False), ('(1,3)', False), ('(1,4)', True), ('(2,1)', True), ('(2,2)', False), ('(2,3)', False), ('(2,4)', True), ('(3,1)', True), ('(3,2)', True), ('(3,3)', False), ('(3,4)', True), (
'(4,1)', False), ('(4,2)', True), ('(4,3)', False), ('(4,4)', True), ('1h(1,1)', False), ('1h(1,2)', False), ('1h(1,3)', False), ('1h(1,4)', False), ('1h(2,1)', False), ('1h(2,2)', False), ('1h(2,3)', False), ('
1h(2,4)', False), ('1h(3,1)', False), ('1h(3,2)', False), ('1h(3,3)', False), ('1h(3,4)', False), ('1h(4,1)', False), ('1h(4,2)', True), ('1h(4,3)', False), ('1h(4,4)', False), ('1horizontal', True), ('1s(1,1)',
 False), ('1s(1,2)', False), ('1s(1,3)', False), ('1s(1,4)', False), ('1s(2,1)', False), ('1s(2,2)', False), ('1s(2,3)', False), ('1s(2,4)', False), ('1s(3,1)', False), ('1s(3,2)', True), ('1s(3,3)', False), ('1
s(3,4)', False), ('1s(4,1)', False), ('1s(4,2)', False), ('1s(4,3)', False), ('1s(4,4)', False), ('1size2', True), ('1size3', False), ('1size4', False), ('1v(1,1)', False), ('1v(1,2)', False), ('1v(1,3)', False)
, ('1v(1,4)', False), ('1v(2,1)', False), ('1v(2,2)', False), ('1v(2,3)', False), ('1v(2,4)', False), ('1v(3,1)', False), ('1v(3,2)', False), ('1v(3,3)', False), ('1v(3,4)', False), ('1v(4,1)', False), ('1v(4,2)
', False), ('1v(4,3)', False), ('1v(4,4)', False), ('1vertical', False), ('2h(1,1)', False), ('2h(1,2)', False), ('2h(1,3)', False), ('2h(1,4)', False), ('2h(2,1)', True), ('2h(2,2)', False), ('2h(2,3)', False),
 ('2h(2,4)', False), ('2h(3,1)', True), ('2h(3,2)', False), ('2h(3,3)', False), ('2h(3,4)', False), ('2h(4,1)', False), ('2h(4,2)', False), ('2h(4,3)', False), ('2h(4,4)', False), ('2horizontal', True), ('2s(1,1
)', True), ('2s(1,2)', False), ('2s(1,3)', False), ('2s(1,4)', False), ('2s(2,1)', False), ('2s(2,2)', False), ('2s(2,3)', False), ('2s(2,4)', False), ('2s(3,1)', False), ('2s(3,2)', False), ('2s(3,3)', False),
('2s(3,4)', False), ('2s(4,1)', False), ('2s(4,2)', False), ('2s(4,3)', False), ('2s(4,4)', False), ('2size2', False), ('2size3', True), ('2size4', False), ('2v(1,1)', False), ('2v(1,2)', False), ('2v(1,3)', Fal
se), ('2v(1,4)', False), ('2v(2,1)', False), ('2v(2,2)', False), ('2v(2,3)', False), ('2v(2,4)', False), ('2v(3,1)', False), ('2v(3,2)', False), ('2v(3,3)', False), ('2v(3,4)', False), ('2v(4,1)', False), ('2v(4
,2)', False), ('2v(4,3)', False), ('2v(4,4)', False), ('2vertical', False), ('3h(1,1)', False), ('3h(1,2)', False), ('3h(1,3)', False), ('3h(1,4)', False), ('3h(2,1)', False), ('3h(2,2)', False), ('3h(2,3)', Fal
se), ('3h(2,4)', True), ('3h(3,1)', False), ('3h(3,2)', False), ('3h(3,3)', False), ('3h(3,4)', True), ('3h(4,1)', False), ('3h(4,2)', False), ('3h(4,3)', False), ('3h(4,4)', True), ('3horizontal', True), ('3s(1
,1)', False), ('3s(1,2)', False), ('3s(1,3)', False), ('3s(1,4)', True), ('3s(2,1)', False), ('3s(2,2)', False), ('3s(2,3)', False), ('3s(2,4)', False), ('3s(3,1)', False), ('3s(3,2)', False), ('3s(3,3)', False)
, ('3s(3,4)', False), ('3s(4,1)', False), ('3s(4,2)', False), ('3s(4,3)', False), ('3s(4,4)', False), ('3size2', False), ('3size3', False), ('3size4', True), ('3v(1,1)', False), ('3v(1,2)', False), ('3v(1,3)', F
alse), ('3v(1,4)', False), ('3v(2,1)', False), ('3v(2,2)', False), ('3v(2,3)', False), ('3v(2,4)', False), ('3v(3,1)', False), ('3v(3,2)', False), ('3v(3,3)', False), ('3v(3,4)', False), ('3v(4,1)', False), ('3v
(4,2)', False), ('3v(4,3)', False), ('3v(4,4)', False), ('3vertical', False)]
   1 2 3 4
A  0 0 0 -
B  - - 0 0
C  - - - -
D  0 0 0 0

Computed solution in 0.0913 seconds
```

5 x 5 – 3 Ships


Grid size= 5
Ships count= 3
Ship #1: size2
Ship #2: size3
Ship #3: size4
Computed solution in 0.1763 seconds

6 x 6 – 3 Ships


docker exec -it b34b78bcf053fbde48b4c5ca0059e9ad8b42be803a3a0c39ec9725503a217ba3 /bin/sh
Ships count= 3
Ship #1: size2
Ship #2: size3
Ship #3: size4