# **COMP7018 Report**

[**COMP7018 Report**](#_jc9mwnsdnime) **1**

[**1. Introduction**](#_f65j1nd3vomq) **2**

[**2. Heuristic agent**](#_qr77mabx4osh) **2**

[2.1 Ship placement heuristic](#_l41f5oh92uc0) 2

[2.2 Hit heuristic](#_qtdyh423fm15) 4

[2.3 Attempts heuristic](#_r2geey5eq3r8) 5

[**3. Testing the heuristic agent**](#_qm0twi9xt128) **10**

[**4. Iterative Deepening with Minimax**](#_lvn79a2ba2x8) **13**

[**5. Iterative deepening with minimax performance**](#_9odxrastyuc) **16**

[**6. Logic**](#_dfad7i3u6479) **18**

[6.1 Predicate Logic for Gunboats](#_8jg7ortfthr0) 18

[6.2 Describing an instance with the rules](#_awc8hixeof4) 20

[6.3 Proof by Resolution](#_azkeomvidk6k) 21

[**7. Probabilistic reasoning**](#_7ofdlz7kjy2v) **23**

[**8. Research papers**](#_y1axw9hflh6s) **28**

[8.1 The ethics of algorithms: Mapping the debate](#_1xh5e63iijlh) 28

[8.2 Human-Aligned Artificial Intelligence is a Multiobjective Problem](#_ec2euxnse0nr) 28

[8.3 Ethical concerns in rescue robotics: a scoping review](#_p1gmn3s373s3) 29

[8.4 Logic-Based Technologies for Intelligent Systems: State of the Art and Perspectives.](#_qwpq2623y8yt) 29

[**9. Conclusion**](#_zhyj6xvxyj0s) **31**

[**References**](#_v22t74e558lx) **32**

# 1. Introduction

Artificial Intelligence (AI) is one of the fastest growing sectors of the IT industry, AI at the core is the method of bestowing rational decision making in a machine. To provide a machine the ability to make the “correct” decision regarding provided data, often considered the founder of AI Alan Turing designed a method called the Turing Test. The Turing Test requires a computer to successfully answer a series of questions from a human interrogator without being recognised as a computer, Alan Turing proposed that this is when “true” AI has been successfully developed.

In this project three types of AI have been designed and implemented to play against one another in a game of “Gunboats”; Random, Heuristic and Minimax. All three AI efficiency is tested through a series of experiments which require each to play against one another for 100 games. After documenting and analysing the results a hypothesis can be made. The aim of this project is to learn, design and experiment with AI methods to come to a conclusion of what method is better suited for “Gunboats”.

Gunboats is played with five ships of length: 5, 4, 3 and two 2 celled ships. Therefore the player is required to place one ship at the beginning of the first five turns until all five ships are placed. The Random AI method does this by selecting a random cell for the ship start and in a random horizontally or vertical direction in the next *n* cells selects the end ship cell, in this case *n* depends on a ships size. After a ship is placed the player is then able to make an attempt at hitting an enemy ship of which the position is unknown.

In this case Random AI (the player) will select a random cell from a list of available cells for attempts, if the cell is a valid attempt, the attempt is made and the cell is removed from available cells. However, whether the attempt is a hit or miss, the next attempt by Random AI is a random cell of currently available attempts, this is due to the fact that previous hits are not considered by Random AI. This means that no logic is implemented in this AI method.

# 2. Heuristic agent

The heuristic AI uses 3 different heuristics, for ship placement, for attempts and to reason with hits. To start a turn, the first heuristic used is for ship placement.

### 2.1 Ship placement heuristic

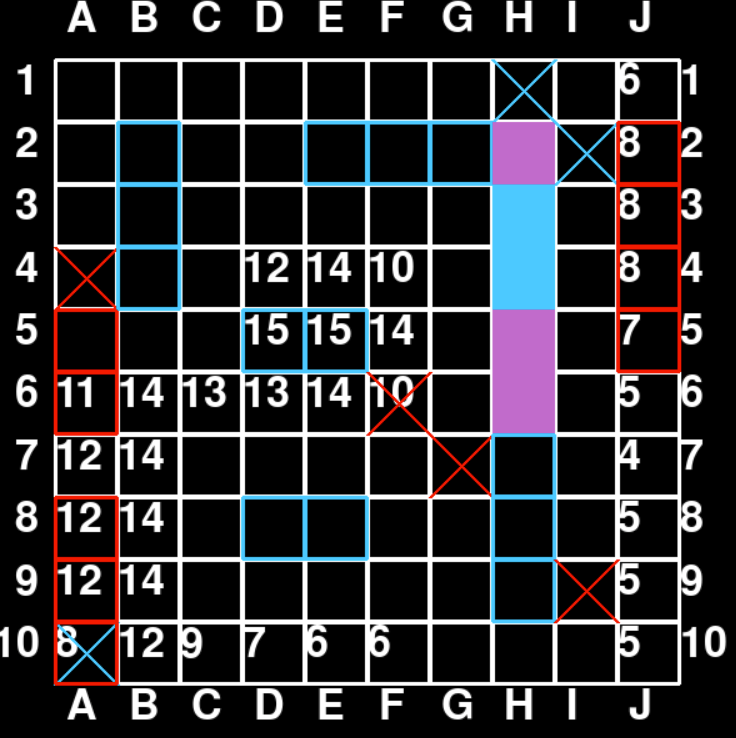
*For python implementation see heuristic.py.*

In order to place a ship, we first calculate the length of the ship that is needed. We calculate the available ship spaces for ship placement after removing our own ship spaces, previous attempts and crashes. The heuristic AI then loops through all of the available cells in the board and calculates the cell score as follows:

**cell\_score** = the number of available spaces within the 1 square boundary of the cell

*Note*: This score has a range between 0 and 8 inclusive.

The valid ships from this cell are then calculated. If there are none, we will continue to the next cell. We then look at the end cell of each possible ship and calculate the cell\_score for this cell. If we find a cell\_score and ship\_score that add to 16, we will select this range of cells to place a ship. Since the order of the board is shuffled before looping through the cells, this allows us to randomly select an “optimal” choice without needing to necessarily loop through all of the cells in the board.

In the example below in figure 1, we can see the blue player as the heuristic AI. We can see the associated scores that have been calculated for each cell based on the aforementioned ship heuristic. All of the possible cells where a ship placement is possible from has been calculated as there is no score of 16. In this instance, the cells in D5 and E5 were selected for the length 2 ship as these have the highest cell score. It is worth noting that the reason the score is 15, and not 16, is because C4 is not an available attempt as it is adjacent to a blue ship and therefore is not counted as “free”. This helps to further space out ships across the board to decrease uncertainty. It is worth noting that we are not concerned about the double counting of cells as this will be consistent across all ship scores and will not affect the overall heuristic logic. 

*Figure 1: Ship Heuristic*

The reason that this heuristic was selected was in order to reason with the uncertainty on the board, and use the ships to systematically reduce this uncertainty. Through selecting squares where we have no other information, we vastly improve our chances of a crash, thereby giving both ourselves, and our opponents, more information about the board. A possible improvement not implemented would be to only use this heuristic when the player is winning (as determined by another heuristic). Else, if the opponent were to play correctly, in trying to find more enemy ships through crashes, you would likely lose the game.

### 2.2 Hit heuristic

*Python implementation can be seen in heuristic.py (function: heuristic\_attempt)*

The first time that an enemy ship is hit, this gets written into the board.ships\_other\_player. This maintains an idea of the other players' ships from the player’s point of view.

The first thing the next heuristic does is to loop through the other player’s ships to see if any of the surrounding horizontal and vertical squares could also be a hit. When there is just one hit on a ship, a score of 1 gets applied to the valid horizontal and vertical squares a distance of 1 away from the other player’s ships. If there are multiple hits calculated, then a score of 1 gets applied to the valid squares in the hit direction of the ship (vertical or horizontal).

In the below instance, where there is just one red hit on a blue ship, the calculated scores are in all horizontal and vertical squares a distance of 1 square away from the hit at (5,5).

|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 |  |  |  |  |  |  |  |  |  |  |
|  | 2 |  |  |  |  |  |  |  |  |  |  |
|  | 3 |  |  |  |  |  |  |  |  |  |  |
| Y | 4 |  |  |  |  | 1 |  |  |  |  |  |
|  | 5 |  |  |  | 1 | X | 1 |  |  |  |  |
|  | 6 |  |  |  |  | 1 |  |  |  |  |  |
|  | 7 |  |  |  |  |  |  |  |  |  |  |
|  | 8 |  |  |  |  |  |  |  |  |  |  |
|  | 9 |  |  |  |  |  |  |  |  |  |  |
|  | 10 |  |  |  |  |  |  |  |  |  |  |

*Figure 2 - Hit heuristic with four possibilities*

After the next attempt being made at (5,4), the board state is as follows, with scores of 1 at (5,3) and (5,6) and all other scores 0 (not shown for clarity). The reason that (5,4) and (5,5) do not have a score of 1 is that they are not valid squares and therefore not considered for scoring. Through following this process, we will always be able to “sink” a ship after having at least one hit on it.

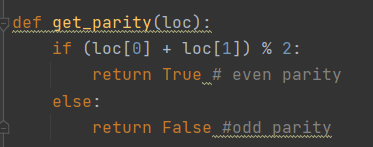
|  |  |  |  |  | X |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 |  |  |  |  |  |  |  |  |  |  |
|  | 2 |  |  |  |  |  |  |  |  |  |  |
|  | 3 |  |  |  |  | 1 |  |  |  |  |  |
| Y | 4 |  |  |  |  | X |  |  |  |  |  |
|  | 5 |  |  |  |  | X |  |  |  |  |  |
|  | 6 |  |  |  |  | 1 |  |  |  |  |  |
|  | 7 |  |  |  |  |  |  |  |  |  |  |
|  | 8 |  |  |  |  |  |  |  |  |  |  |
|  | 9 |  |  |  |  |  |  |  |  |  |  |
|  | 10 |  |  |  |  |  |  |  |  |  |  |

*Figure 3 - Hit heuristic with two possibilities*

### 2.3 Attempts heuristic

*Python implementation seen in heuristic.py (function: heur\_attempt)*

If no possible squares are returned that could be hits, then the next heuristic is then used instead. The heuristic first calculates the available attempts given the boardstate. Then, the parity is the calculated for each square as follows:



*Figure 4 - Calculating cell parity*

Since the minimum length of a ship is 2, we can select squares with either an odd or even parity. This guarantees that with half the number of squares that we will at least be able to hit every ship, and then hunting out the rest of the ship is taken care of by the hit heuristic.

After calculating the parities, we will return the available attempts as either all odd, or all even, depending on which set has less elements.

We will then loop through these possible attempts and calculate a heuristic as follows:

The scores get initially set as 0 (although this number is arbitrary). The horizontal and vertical boundaries are calculated for a cell. If a boundary is a hit for that player, then:

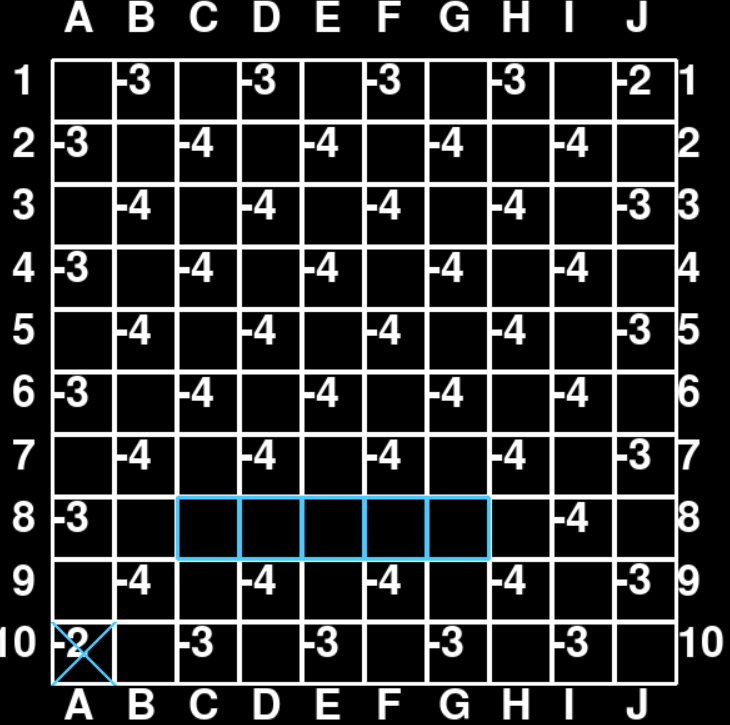
cell\_score[i] = cell\_score[i] + 1

Otherwise, if the cell is in the board, then cell\_score[i] = cell\_score[i] - 1. This will output a list of cells and scores, where the higher the score, the more likely a surrounding hit. A low score indicates that there is free space around the cell, and therefore less known.

For the first 5 turns when not all of the ships are laid, we want to avoid a particular game mechanic where if an attempt is made on a cell and then an enemy ship is laid on top of this, we cannot win the game. In order to avoid this, for the first 5 attempts, we want to choose the highest score possible (remembering that if we have a hit in the first 5 turns, then the hit heuristic will be used). That is to say, we will be picking the corner squares where probabilistically there are the least number of possible ships (2). This is a workaround to avoid the instant loss condition.

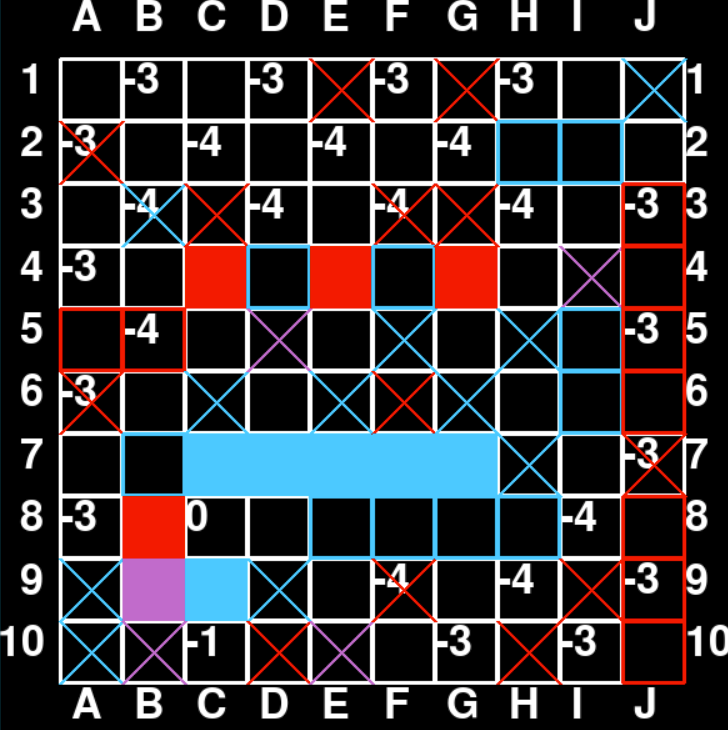
Past turn 5, we want to pick the lowest score from this heuristic. This allows us to make attempts where we currently know least.

An example of this heuristic in action can be seen below. Below, we can see the initialisation of the board attempts, where we choose the highest score of -2 as this is before turn 5.



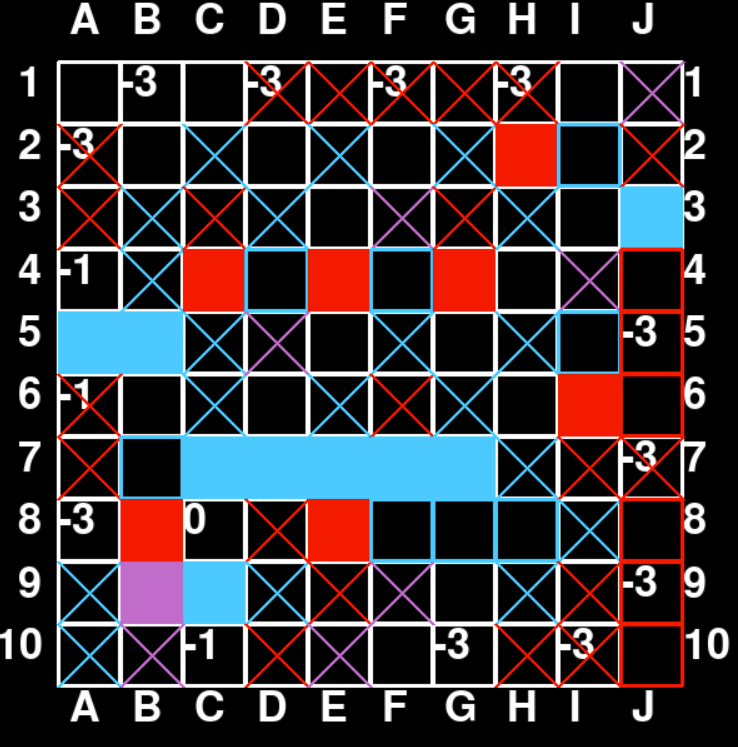
*Figure 5 - Attempt Heuristic at the start of a game*

Fast forward to being in the middle of a game and we can see that a lot of the middle squares have already been selected and the scores are now being favoured on the outer edges of the board. We will continue to make some incorrect guesses in the upper regions of the board until the -3’s becomes the lowest score.



*Figure 6 - Attempt heuristic in the middle of a game*

We can now see that all of -4’s are now hits and we have eventually gotten a hit on the outer edge at J3.



*Figure 7 - Attempt Heuristic in the end game*

# 

# 3. Testing the heuristic agent

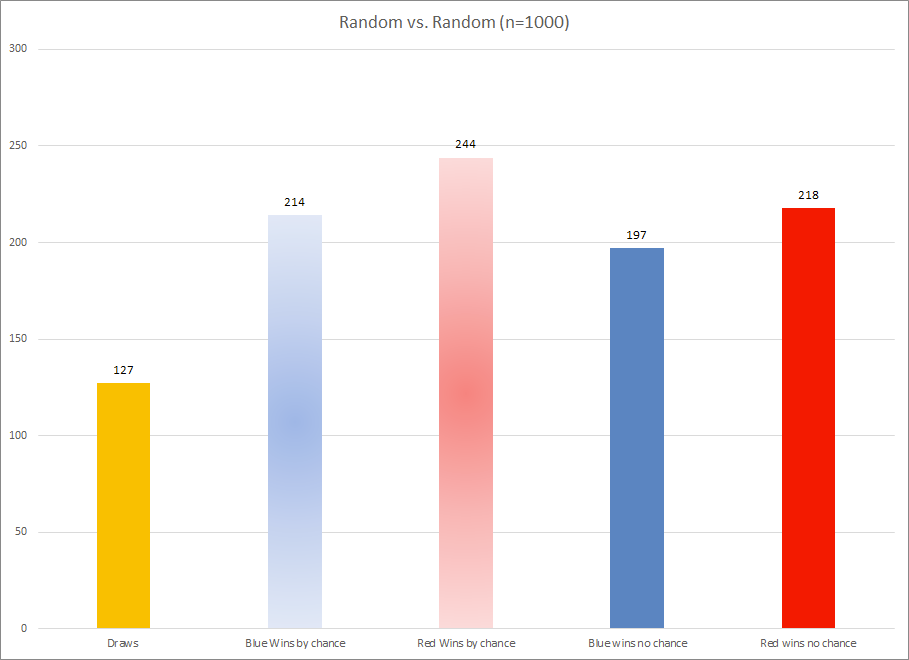
Due to the game mechanic where misses stay as misses, the outcome of a game was recorded as one of 3 different types:

**Win without chance** - One player sinks all of the other players ships and neither player had the other player’s ship placed over a previously placed attempt.

**Win by chance** - One player made an attempt and the other player placed a ship over that attempt, meaning that the other player cannot win.

**Draw** - Both players make attempts where the other player places a ship over each of the respective attempts. This means that no player can win.

Firstly, we want to investigate what “normal” looks like when playing two random agents against each other. This both confirms that the simulations are being run correctly and gives us a baseline to compare the heuristic agent against. 1000 games were simulated with the random agents and the results can be seen below:



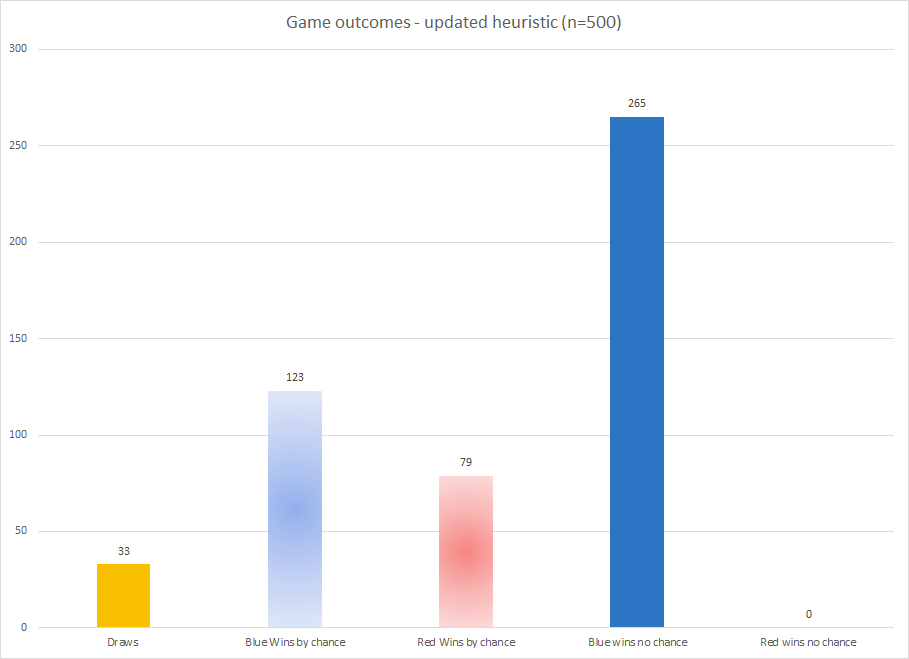
*Figure 8 - 1000 games of Random vs. Random*

What is noteworthy about these results is that the blue player wins slightly less than the red player. This is likely due to the fact that the blue player’s first attempt can only disadvantage them due to the game mechanic of misses staying as misses. Additionally, this attempt has a 0% chance of hitting an enemy ship, thereby giving an advantage to the red player from the start. When running simulations for the heuristic agent against the random agent, we will therefore make the heuristic the blue player such that it has a slight disadvantage. This means that any results we do find are even more significant.

500 games were run with the heuristic AI playing as Blue against the random AI as red. There are multiple levels that we can define success for the Heuristic AI over the random AI.

The first level is that the Heuristic has more outright wins than the random AI. For our heuristic AI, we managed to achieve this 100% of the time. In our 500 games, 265 of the games the heuristic won without chance. This is in comparison to 0 games won by the random player. It is clear at this instance, with the disadvantage of playing as the blue player, the heuristic wins every time when we do not consider the “misses” game mechanic.

The second level for success is having more “wins by chance” than the random player. We do this through exploiting the “misses” game mechanic. For the first five turns, we avoid placing attempts in the high probability of ship placement squares (i.e. the centre) and instead favour the edges of the board. This strategy can only be employed when playing against a random AI as it is exploiting the probabilities of possible ship placements. We would expect there to be less than a 50% win rate playing as blue in a random versus random game in this. In the below graph, we can see that there are 123 wins by chance for the heuristic in comparison to 79 for the random player. This is significantly higher for the heuristic at 60.08% compared to 39.92% for the random agent.

*Figure 9 - 500 games of heuristic (blue) vs. random (red)*

Thus, we can conclude that the heuristic is performing better than the random agent on two different measures, as well as having an overall win percentage of 77.6%.

The tests can be run by running view.py in the “master” branch. The resulting outputs will be written into a .xlsx file in your working directory.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 4. Iterative Deepening with Minimax

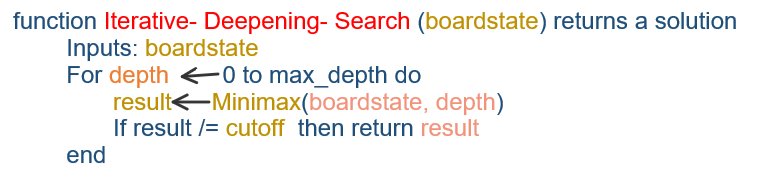
**Iterative deepening** is a searching algorithm in AI. It is an improved version of depth limited search (DLS) where it executes the DLS from 0 to max\_depth.The properties of Iterative deepening depth first search (IDDFS) include:

Time complexity : O(b^d)

Space complexity : O(bd)

Where d is depth and b is the branching factor.

Below, in figure 10, is the pseudocode used to implement the iterative deepening search with minimax:



*Figure 10 - Iterative Deepening Search pseudocode*

**Minimax** is a game theory algorithm that finds the best possible move for a player. It is a backtracking method that is commonly used in two-player games for decision-making. One player is the maximizer, while the other is the minimizer. The maximizer always seeks to increase the game score, while the minimizer always wants to decrease it. The whole game state must be available to execute the minimax, and it is expected that the minimizer will always play optimally, thus the states are chosen accordingly.

**Implementation of minimax for Gunboat game**

The assumptions we made to implement minimax for the gunboat game are:

* The complete gamestate information is available
* Blue player is the maximising player and red is the minimising player
* The game score will increment by 1 if a red ship cell is hit by blue player
* The game score will be decremented by 1 if a blue ship is hit by red player
* The score for all other moves is 0
* Minimax is calculated only for attempts

How our Minimax logic will work for a random boardstate is explained below.

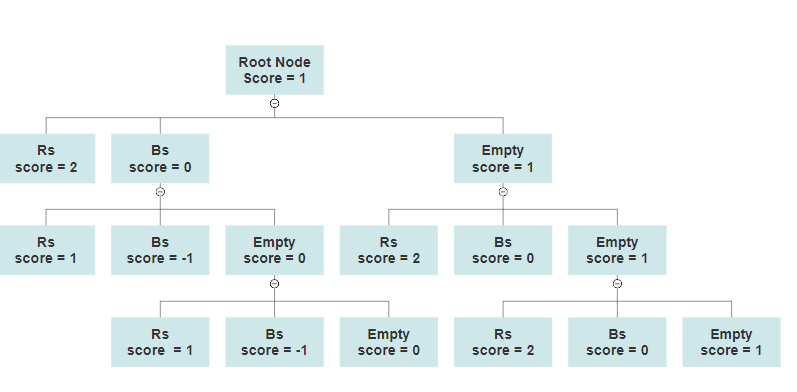
boardstate =[['-', '-', '-', 'Bs', 'Bs', 'Rh', 'Rh', '-', '-', '-'], ['Rh', '-', 'Bm', '-', 'Bh', 'Bh', 'Bh', 'Bh', 'Rs', 'Rh'], ['Rh', '-', '-', '-', 'Bm', 'Rh', '-', '-', '-', 'Rh'], ['Rh', '-', '-', 'Rm', 'Rm', 'Rh', '-', '-', '-', 'Rh'], ['Rh', 'Bh', 'Bh', '-', '-', '-', '-', '-', '-', '-'], ['Rh', '-', '-', '-', 'Rm', '-', '-', '-', '-', '-'], ['-', 'Bh', 'Bh', '-', 'Rm', '-', '-', '-', '-', 'Bh'], ['-', '-', '-', 'Bm', '-', '-', 'Bm', '-', '-', 'Bh'], ['-', '-', '-', 'Bh', 'Bh', 'Bh', 'Bh', '-', '-', 'Bh'], ['Rh', 'Rh', '-', '-', '-', '-', '-', '-', '-', '-']]

# Blue ship(Bs), Red ship(Rs), Blue hit(Bh), Red hit(Rh) , Blue miss(Bm), Red miss(Rm), Crash(C), Red and Blue miss (RBm)

If given above is the board state we can extract the following information from that

* Available blue ship cells - 2
* Available red ship cells - 1
* Empty cells - many
* Current\_score = blue hits - red hits = 1

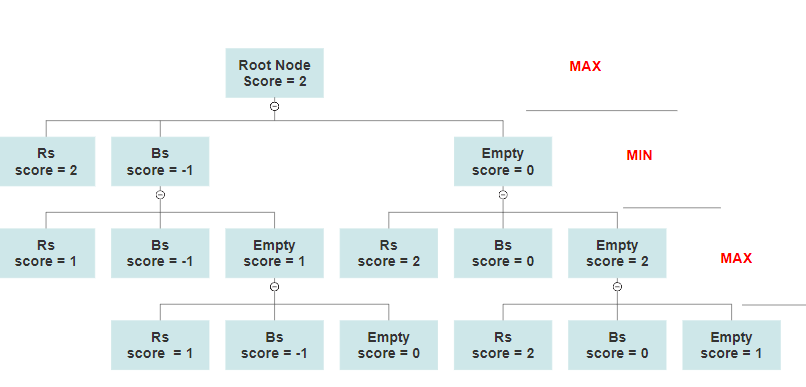
To run minimax, we'll require the game's decision tree, which includes all possible future moves and corresponding boardstate, as well as the leaf node score at the given depth. We calculate the nodes and their values for depth = 3 in order to show the minimax concept.



*Figure 11 - Decision tree constructed to get the leaf node score for depth=3*

The current boardstate is represented by the root node score, which is the game's current score. The game score for the next level board states is dependent on the potential future moves. The current player is the maximum player, and he can attempt a Red ship cell, a Blue ship cell, or any other vacant cell. If he chooses the redship cell, the game score will increase by 1, if he chooses the blue ship cell, the score will fall by 1, and if he chooses the empty cell, the score will remain the same. Similarly, the score is calculated until depth 3 is reached, and the leaf node scores are obtained.

Now that we have the leaf node scores we can run the minimax algorithm. The below tree gives the node scores after minmax is executed.



*Figure 12 - minimax tree for depth 3*

Minmax is a backtracking method that starts at the leaf node, as previously stated. In the example, it is the maximum player's turn, and he will choose the move that will award him the most points. After that, it's the turn of the min player, who must choose the move with the lowest score. Then it's the turn of the maximum player, and his decision is the optimal move for that boarstate.

Looking at the minimax tree, we can deduce that the max player (the blue player) would try the red ship cell, raising the game score to 2.

This logic's implementation is done in Python, and it has been verified to work.

# 5. Iterative deepening with minimax performance

Experiments of minimax playing versus a random AI player and a heuristic player are used to evaluate minimax performance. The depth is varied, and the results are summarised here.

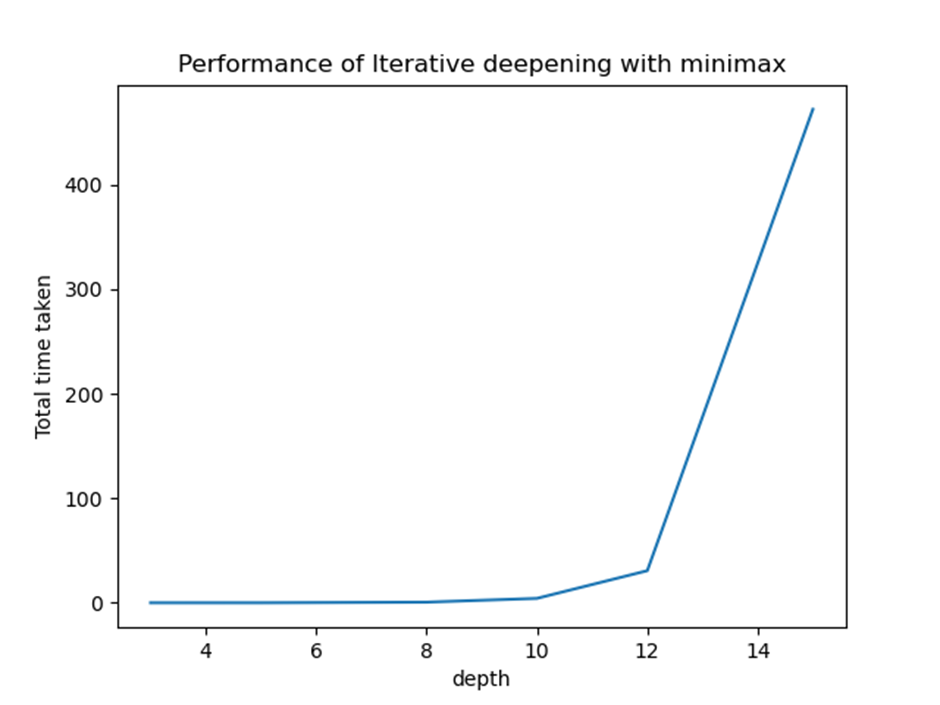
*Table 1 - Performance of minimax for different depths*

| Depth | Total number of nodes evaluated | Time taken for the minmax logic to run |
| --- | --- | --- |
| 1 | 1 | 0.00001 sec |
| 2 | 4 | 0.001 sec |
| 3 | 13 | 0.0019 sec |
| 4 | 40 | 0.005 sec |
| 5 | 121 | 0.019 sec |
| 6 | 364 | 0.038 sec |
| 7 | 1091 | 0.11 sec |
| 8 | 3248 | 0.32 sec |
| 9 | 9551 | 0.95 sec |
| 10 | 27564 | 2.74 sec |

*Table 2: Performance of Iterative deepening with minimax for different depths*

| Depth | Total number of nodes evaluated | Time taken for iterative deepening with minimax to run |
| --- | --- | --- |
| 3 | 18 | 0.003 sec |
| 5 | 179 | 0.02 sec |
| 8 | 4916 | 0.59 sec |
| 10 | 41997 | 4.2 sec |
| 12 | 331032 | 30.7 sec |
| 15 | 5633724 | 472.1 sec |

The number of nodes to be assessed and the time it takes to evaluate all the nodes increases dramatically for both minimax and iterative deepening with minimax. IDDFS will execute the minimax from depth 0 to the maximum depth, so more nodes are assessed for iterative deepening with minimax for the same depth.



*Figure 13 - Performance of Iterative deepening with minimax*

The graph above shows the time it takes to conduct iterative deepening with minimax for different depths. We can observe that after depth 10, the time taken increases exponentially, but before that it increases slowly.

# 6. Logic

When one talks about “logic”, it most often refers to either predicate or propositional logic, depending on the context. First order logic, also known as predicate logic, is used in a number of fields, including mathematics and computer science, to help define logical statements that can be used as a system to describe variables. In propositional logic, true or false propositions can be defined as well as relations between them. Propositional logic does not deal with non-logical objects, a feature which differentiates it from predicate logic. An example of propositional logic, would be the following statement: “the table is made of only wood”. This statement would be true if a table were made of only wood, and would be false for all other materials. The statement cannot be both true and false. An example of a statement that is not propositional logic would be: “what is the table made of?” This is an open question rather than a true-false statement, and therefore is not logic.

Similar to propositional logic, predicates can be used to define a formal system. Predicate, on a foundational level, consists of variables and constants, from which rules can be derived, leading to the creation of a knowledge base. Functions can also be defined and they serve a similar purpose as to what they would in a programming language. For example, let us denote *sqrt(x)* as a function which determines the square root of a positive integer, a variable. We could write sqrt(4), however this would normally be written as √4, or even 41/2. Logic expressions can be thought of as another way of describing operations, statements and relationships.

Moreover, predicate differs from propositional logic syntactically, as well as in its use of *quantifiers*. These are the ∀ and ∃ symbols commonly used in mathematics, as well as their opposites which are used by declaring ¬. These can be read as: “for all” and “there exists” respectively. This allows us to consider instances of variables, and can further be used to deduce implications to describe the relationships.

## 6.1 Predicate Logic for Gunboats

For this assignment, we will be using logic to describe an instance of Gunboats. As seen in Figure 14, a game is taking place where the Red Player has placed their second ship, and is about to place their second attempt. Here, we will be playing as Red and will look to deduce their next move.

|  |  |  |  |  | X |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  | 1 |  |  |  |  |  |  |  |  |  |  |
|  | 2 | X |  |  |  |  |  |  |  |  |  |
|  | 3 |  |  |  |  |  |  |  |  |  |  |
| Y | 4 |  |  |  |  |  |  |  |  |  |  |
|  | 5 |  |  |  |  |  |  |  |  |  |  |
|  | 6 |  |  |  |  |  |  |  |  |  |  |
|  | 7 |  |  |  |  |  |  |  |  |  |  |
|  | 8 |  |  | ~~X~~ |  |  |  |  |  |  |  |
|  | 9 |  |  |  |  |  |  |  |  |  |  |
|  | 10 | ~~X~~ |  |  |  |  |  |  |  |  |  |

*Figure 14 - Initial board state for logic*

1. Define the board as a 10x10 grid in (x,y) coordinates:
   1. ∀x,y,**∈Z s.t.** 0<x,y<11
2. Define players:
   1. Playerr **⇔** ¬Playerb
3. KB = Knowledge Base
   1. Htx,y (Player) A player can have a hit on cell x,y at time t
   2. Sx,y (Player) A players ship on cell x,y
   3. Atx,y (Player) A players attempt on x,y at time t
   4. Ctx,y Both players ships have crashed at that cell at time t
   5. Vtx,y (Player) Valid space for a player
   6. Btx,y (Player) A players ship borders
   7. Ntx,y (Htx,y) One of the horizontal/vertical cells adjacent to a hit is also a hit

Valid attempts which are hits imply the other player’s ship.

Atx,y (Playerr) **^** Vtx,y (Playerr) **^** Htx,y (Playerr) **⇒** Sx,y (¬Playerr) (1.1)

Valid attempts on the other players' ships are hits.

Atx,y (Playerr) **^** Vtx,y (Playerr) ^Sx,y (¬Playerr) **⇒** Htx,y (Playerr) (1.2)

Hits imply attempts on the other players ships.

Htx,y (Playerr) **⇒** Atx,y (Playerr) ^Vtx,y (Playerr)^Sx,y (¬Playerr) (1.3)

Ships for both players imply crashes.

Sx,y (¬Player) **^** Sx,y (Player) **⇔** Ctx,y (1.4)

Ships have boundaries.

Sx, y (Playerr) **⇒** Btx-1,y+1 (Player) **V** Bx+1,y+1 (Playerr) **V** Bx,y+1 (Playerr) **V** Bx+1,y (Playerr) **V** Bx-1,y (Playerr) **V** Bx-1,y-1 (Playerr) **V** Bx,y-1 (Playerr) **V** Bx+1,y-1 (Playerr) (1.5)

Not valid on a cell implies an attempt, crash, ship or boundary at that cell.

¬Vtx,y (Playerr) **⇔** A<tx,y (Playerr) **V** C<=tx,y **V** Sx,y (Playerr) **V** Bx,y (Playerr) ∀x,y,**∈Z s.t.** 0<x,y<11 (1.6)

Not a hit on a cell implies there is not a ship there for the opposing player.

¬Htx,y (Playerr) **⇒** ¬Sx,y (¬Playerr) (1.7)

If you have a hit and one of the adjacent cells is a hit, both hits are on one ship.

Htx,y (Playerr) ^ (Htx+1,y(Player) **V** Htx,y+1(Player) **V** Htx-1,y(Player) **V** Htx,y-1(Player) ) **⇒** Ntx,y (Playerr) (1.8)

If you have a hit and none of the adjacent cells is a hit, then there has to be a hit in one of these cells. This works in every combination of ¬V, one of these Cells must be a hit or all invalid

Htx,y (Playerr) ^ ¬Ntx,y (Playerr) **⇒** (Htx+1,y(Player) **V** Htx,y+1(Player) **V** Htx-1,y(Player) **V** Htx,y-1(Player)) **V** (¬Vtx+1,y(Player) **^** ¬Vtx,y+1(Player) **^** ¬Vtx-1,y(Player) **^** ¬Vtx,y-1(Player)) (1.9)

If hit at x,y and x+1,y and turn is less than 4, x-1,y or x+2,y must be a hit. This works in every combination of ¬V, one of these two Cells must be a hit or both invalid

∀ t<4, Htx,y (Playerr) ^ Htx+1,y(Playerr) **⇒** (Htx-1,y(Playerr) **V** Htx+2,y(Playerr)) **V** **(**¬Vtx-1,y(Playerr) **^** ¬Vtx+2,y(Playerr)) (1.10)

If hit at x,y and x-1,y and turn is less than 4, x+1,y or x-2,y must be a hit. This works in every combination of ¬V, one of these two Cells must be a hit or both invalid

∀ t<4, Htx,y (Playerr) ^ Htx-1,y(Playerr) **⇒** (Htx-2,y(Playerr) **V** Htx+1,y(Playerr)) **V** (¬Vtx-2,y(Playerr) **^** ¬Vtx+2,y(Playerr)) (1.11)

If hit at x,y and x,y-1 and turn is less than 4, x,y+1 or x,y-2 must be a hit.This works in every combination of ¬V, one of these two Cells must be a hit or both invalid

∀ t<4, Htx,y (Playerr) ^ Htx,y-1(Playerr) **⇒** (Htx,y-2(Playerr) **V** Htx,y+1(Playerr)) **V** (¬Vtx,y-2(Playerr) **V** ¬Vtx,y+1(Playerr)) (1.12)

If hit at x,y and x,y+1 and turn is less than 4, x,y-1 or x,y+2 must be a hit.This works in every combination of ¬V, one of these two Cells must be a hit or both invalid

∀ t<4, Htx,y (Playerr) ^ Htx,y+1(Playerr) **⇒** (Htx,y+2(Playerr) **V** Htx,y-1(Playerr) ) **V** (¬Vtx,y+2(Playerr) **V** ¬Vtx,y-1(Playerr)) (1.13)

## 6.2 Describing an instance with the rules

Now, using the defined deduction rules and knowledge base, let us consider a logically description of the game board in Figure 14.

1. Round 1:
   1. S1,1(¬Playerr),S1,2(¬Playerr),S1,3(¬Playerr),S1,4(¬Playerr),S1,5(¬Playerr)
   2. A13,8 (¬Playerr)
   3. ¬H13,8 (¬Playerr)
   4. S2,2(Playerr),S3,2(Playerr),S4,2(Playerr),S5,2(Playerr),S6,2(Playerr)
   5. A11,2 (Playerr)
   6. H11,2 (Playerr)
2. Round 2:
   1. S3,4(¬Playerr),S3,5(¬Playerr),S3,6(¬Playerr),S3,7(¬Playerr)
   2. A21,10 (¬Playerr)
   3. ¬H21,10 (¬Playerr)
   4. S9,1(Playerr),S9,2(Playerr),S9,3(Playerr),S9,4(Playerr)

Round 1 consists of six statements. First, not Player Red (i.e. Blue) places a Ship from (1,1) to (1,5), followed by an Attempt which is not a Hit at (3,8). Next, Player Red places a ship from (2,2) to (6,2), with an Attempt at (1,2) which results in a Hit.

Round 2 follows in a similar fashion, where not Player Red places a Ship, an attempt at (1,10) which is not a hit, and then Player Red places their Ship from (9,1) to (9,4). From the Knowledge Base we have Ntx,y (Htx,y) that states a hit on a cell x,y means that one of the adjacent cells will also be a hit. Therefore, Player Red will consider cells (1,1), (1,3) and (2,2). From rule (1.6) it is implied that a cell containing your own ship is not valid, and therefore cannot be attempted, so Red will only consider (1,1) and (1,3) for their next attempt.

## 6.3 Proof by Resolution

We will now use resolution to formally conclude that a cell not yet tried must be a hit. To do this, we have to demonstrate that the opposite of this statement is impossible. Therefore, let us try to disprove that a cell not yet tried must not be a hit; we will be taking cell (1,3).

**¬H21,3 (Playerr) –THIS IS WHAT WE ARE TRYING TO DISPROVE**

¬Ht1,3 (Playerr) **⇒** ¬S1,3 (¬Playerr) by (1.7)

¬Nt1,3 (Playerr) by (1.8)

By (1.9), we can deduce

(Ht1+1,2(Player) **V** Ht1,2+1(Player) **V** Ht1-1,2(Player) **V** Ht1,2-1(Player)) **V** (¬Vt1+1,2(Player) **^** ¬Vt1,2+1(Player) **^** ¬Vt1-1,2(Player) **^** ¬Vt1,2-1(Player))

S2,2(Playerr) **⇒** ¬V22,2 (Playerr) by (1.6)

¬V20,2 (Playerr) by (1.6)

V21,1 (Playerr) by (1.6)

V21,3 (Playerr) by (1.6)

¬(¬Vt1+1,2(Playerr) **^** ¬Vt1,2+1(Playerr) **^** ¬Vt1-1,2(Playerr) **^** ¬Vt1,2-1(Playerr)) by deduction

(Ht1+1,2(Playerr) **V** Ht1,2+1(Playerr) **V** Ht1-1,2(Playerr) **V** Ht1,2-1(Playerr)) by deduction

As ¬H21,3 (Playerr)

(Ht1+1,2(Playerr) **V** Ht1-1,2(Playerr) **V** Ht1,2-1(Playerr)) by deduction

As ¬V22,2(Playerr), ¬V20,2 (Playerr) and (1.3)

Ht1,1(Playerr)

(Ht1,1+2(Playerr) **V** Ht1,1-1(Playerr) ) **V** (¬Vt1,1+2(Playerr) **V** ¬Vt1,1-1(Playerr)) by (1.13)

¬Vt1,1-1(Playerr) by (1.6)

Ht1,1+2(Playerr) **V** ¬Vt1,1+2(Playerr) by (1.13)

Vt1,1+2(Playerr) by (1.6)

Ht1,3(Playerr)

CONTRADICTION

Hence, as we have shown there is a contradiction, we have proven by resolution that a cell not yet tried must be a hit.

# 7. Probabilistic reasoning

There are many tactics that can be implied in Gunboats, one such method is assessing the probability of possible attempts being successful. This section explains probabilistic reasoning and shows how this method could be implemented for an attempt in an AI heuristic. Probabilities can quickly become complicated and extremely difficult to calculate. First, the initial game setup needs to be taken into account: 100 positions, 5 possible ships, 2 players and 1 attempt every turn. Also, the game rules need to be considered such as: A new ship is placed by each player in the first five turns in order of sizes 5, 4, 3, 2, 2 and a ship can only be placed vertically or horizontally.

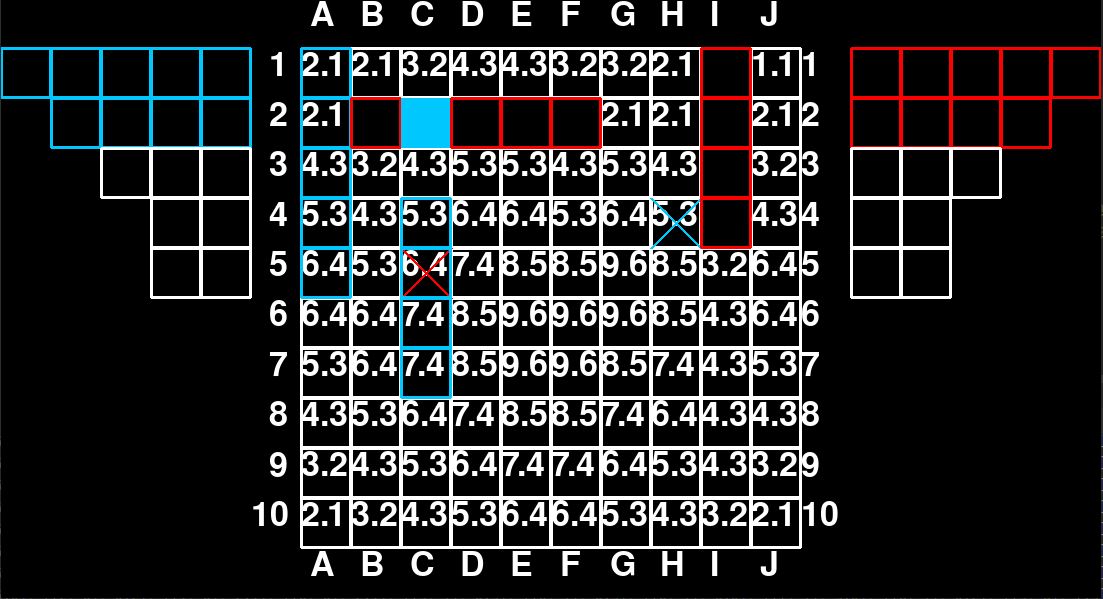
Possibly the most important rule is that a ship can be placed where an attempt has previously been made. In order to calculate these probabilities a separate file “probabilistic\_reasoning.py” was created which requires a board state and player. With this board state it will then calculate the probabilities of each square being a hit for the specified player.

In order to calculate these probabilities, we considered all of the possible ship configurations on the board. This often required considering upwards of 30,000 different board states with different configurations of different length ships. Each time a ship covered a particular square, this square’s score was incremented by 1. After recursing through the different ship possibilities, we divide each cell’s score by the total number of board states considered. This gives us the probability of each cell being a hit.



*Figure 15 - Probabilities of a hit at the start of the game*

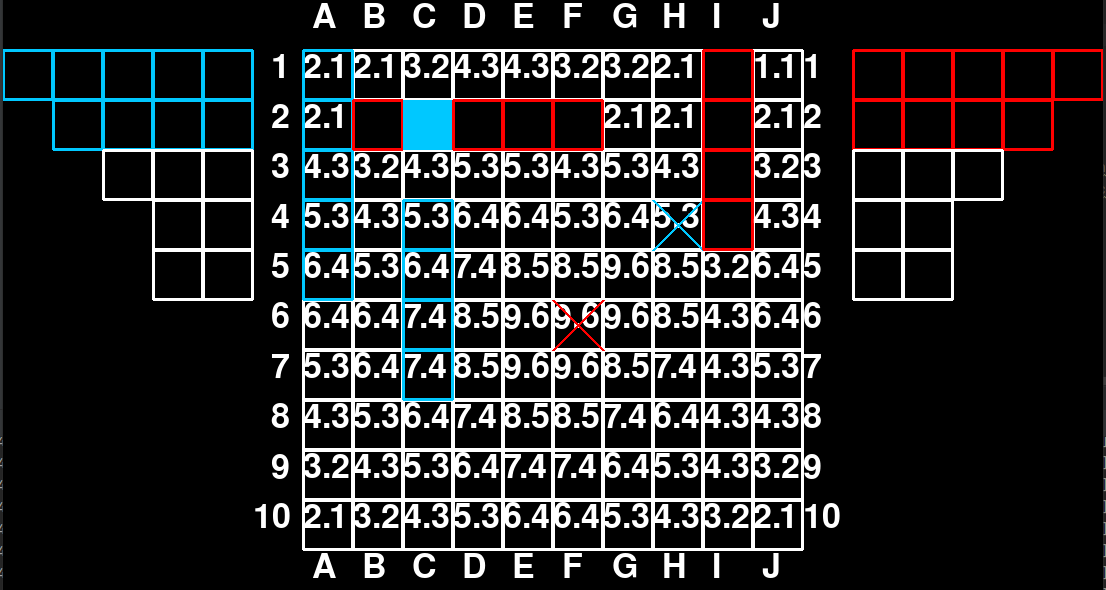
The beginning scenario is shown above in figure 15. As shown, 1 blue and 1 red ship of size 5 have been placed along with one attempt from blue player at position H4. This means this is during the first turn and the red player must next make an attempt. All scenarios will be represented from the red player's point of view and these probabilities show the likelihood of red hitting a ship in each cell. As expected, positions in the centre of the board have a higher probability due to the number of possible placements of the size 5 ship here being greater than edge cells.



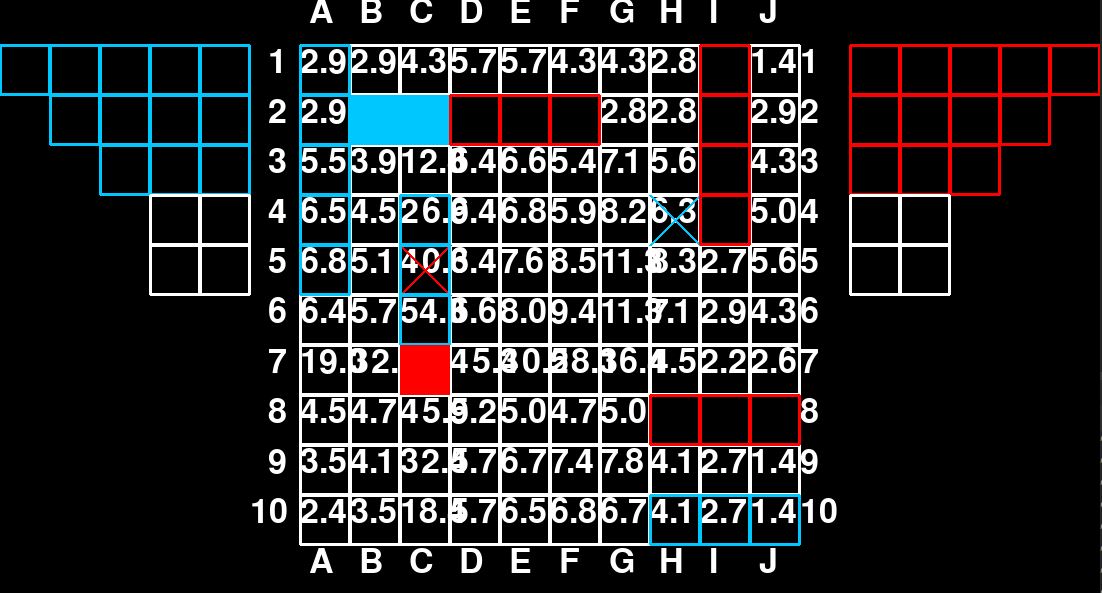
*Figure 16 - Probabilities of a hit - turn 2*

This scene shows that red made an attempt at position C5, this implies no ship is at this position which would dramatically reduce the probability. After this attempt another ship has been placed therefore this position is still considered a possible ship position, as displayed by figure 17 below the probability is still rather high. Red player has also placed another ship of size 4 in the positions I1 - I4, because of this the probability of a blue ship being in the cells around it has decreased slightly.

However as can be seen the probabilities across the board have generally increased, as positions containing red ships are considered invalid and not included in the probabilities calculation. Also as display blue hits or attempts are not considered, although one method to improve these probabilities would be to consider the notion that position containing a blue ship is an invalid attempt for blue player. Implementing this notion into calculations would decrease the probability of a cell containing a blue attempt as well as the cells around it.

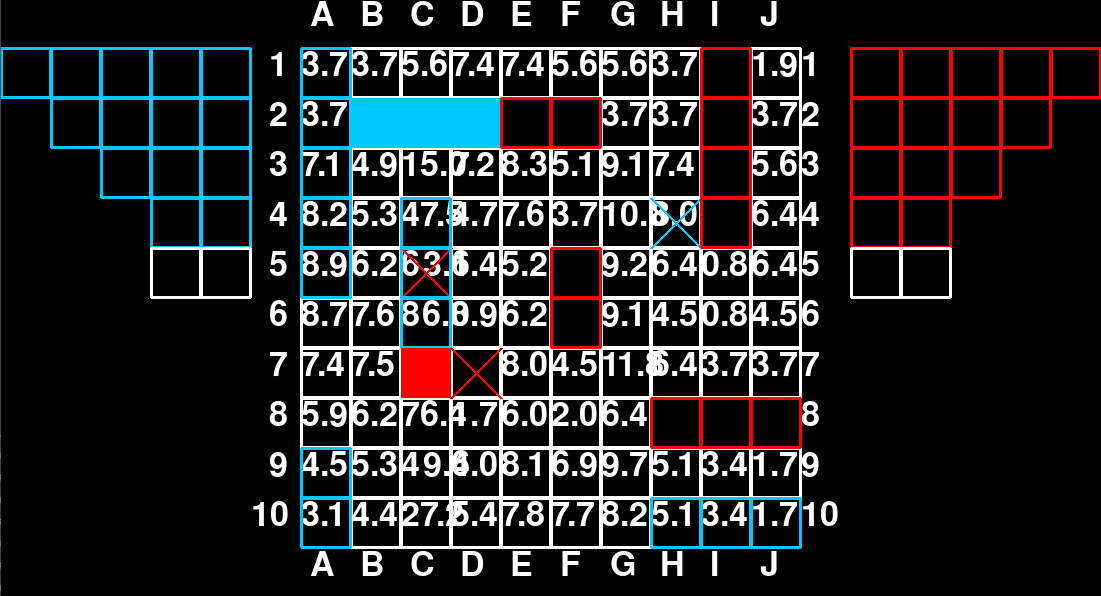


*Figure 17 - Alternative attempt made in turn 2*



*Figure 18 - Probabilities of a hit in turn 3*

In figure 18 above, the red player successfully hit a blue ship in position C7 during the last attempt. Now a hit has been made the focus will be on achieving another successful hit on this ship, also it can be deduced that this ship must be a size of 4 or 5 as size 3 ships had yet to be placed. This successful hit increases the probabilities vertically and horizontally around this cell dramatically as a ship must be here and due to the rule stated above the orientation must be one of the two.



*Figure 19 - Probabilities of a hit in turn 4*

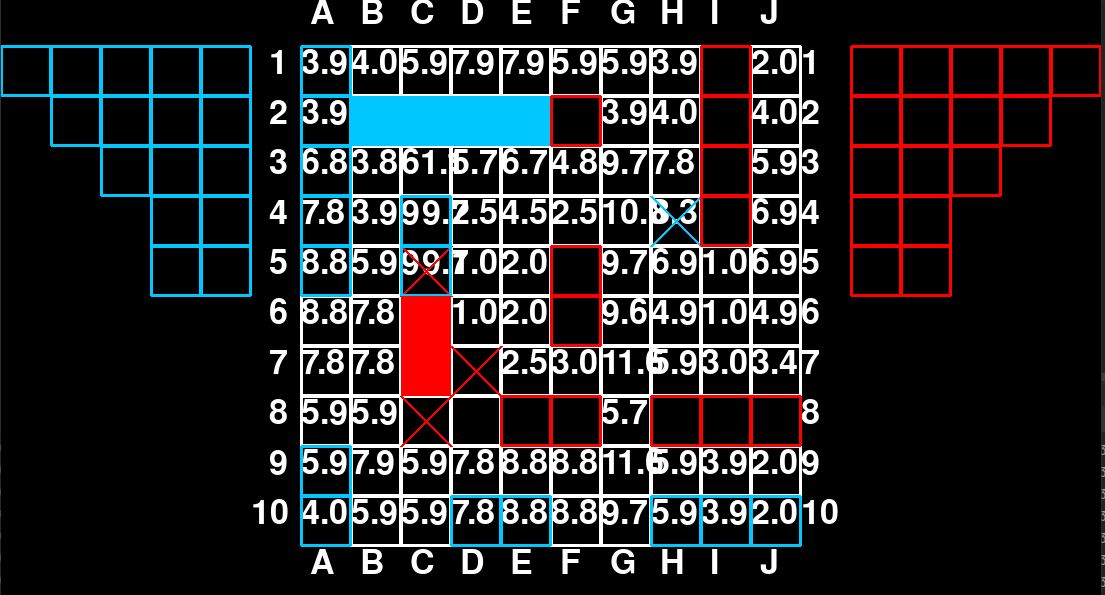
Figure 19 shows that player red’s next attempt was a miss at D7, despite not all ships being placed at the time of this attempt this position is now discarded as a possibility. As stated earlier the probability calculations are based on the fact that only size 4 and 5 ships were on the board at the time of red players hit. Therefore a position in which a miss is made after the hit will be discarded, as neither required ships could be in this position.

In the figure above, an attempt is made at D7, which is a miss. The probabilities for C8, C6 and C5 increase dramatic



*Figure 20 - Probabilities of a hit in turn 5*

The next attempt at C8 by the red player establishes that the blue ship must be in cells C6, C5, and C4. This is shown by the sharp increase to over 99% probability that a ship is in these cells, proving that the red miss previously is no longer a miss and that player red can no longer win, this is made undisputable in figure 21 below. Also position D8 is also considered invalid after this attempt, positions D7 and C8 are miss’s and E8 contains a red ship. Therefore it is impossible for a ship sized 4 or 5 to reach this cell in any possible placement, because of this the cell is discarded as a possibility

*  
Figure 21 - Probabilities of a hit in turn 6*

# 8. Research papers

### 8.1 The ethics of algorithms: Mapping the debate

Jayalakshmi Vijayan

Technology has become a vital component of our daily lives. Artificial intelligence is becoming increasingly important; our behaviours, interests, and preferences are all monitored by data-driven algorithms. The question of ‘how safe the algorithms are’ is crucial. It is difficult to assess the influence of algorithms for a variety of reasons, but we must do so.

Algorithms are data-driven, which can lead to biased or discriminating results. Any decision-making algorithm reflects the designer's value and preferences to some extent. Deindividualization may also occur as a result of algorithmic decision-making. Decisions are made based on group features, so treating and judging persons based on shared attributes is unjust.

Transparent and traceable algorithms are desirable. Accessibility and comprehensibility of information are the two most important aspects of transparency. Information about the functionality of algorithms is difficult to get by, and even when it is, traceability is a problem. They are extremely complex and difficult to comprehend, making it difficult to troubleshoot if an issue arises. It's similar to a black box solution in that we obtain output for a certain input but have no way of knowing why that output occurred.

Another issue is who bears moral and legal responsibility for any algorithmic failure. It is difficult to say who should be held responsible in the case of AI. If the code was explicitly developed by someone, they can be held responsible, but this is not conceivable in the case of self-learning algorithms.

### 8.2 Human-Aligned Artificial Intelligence is a Multiobjective Problem

Methuselah Singh

In their paper *Human-Aligned Artificial Intelligence is a Multiobjective Problem*, Vamplew et al. discuss the concept of “alignment”, meaning an AI system which is developed such that it behaves in a manner which is beneficial to humanity, or rather, its intentions are aligned with human interests (Soares and Fallenstein, 2014). The report identifies and examines the benefits and drawbacks of different types of alignment frameworks. Some examples of these frameworks are legal, military and safety frameworks, which could utilise different ethical ideologies such as utilitarian and deontological.

Utilitarianism goes by the notion that actions should be judged by their consequences, and that some utility metric can be defined such that morally right actions can lead to greater utility value. Another approach would be a legal alignment framework, whereby a system would be developed so as to comply with the legal structure of a certain society.

It is important to consider the wider societal context that AI systems can be implemented in. For instance, we as humans are accustomed to vague terminology that we can use to gauge and infer meaning based on the context, something a programme is unable to do. To ensure consistency across the industry, official bodies should establish a framework that considers an ethical approach during the development of systems.

### 8.3 Ethical concerns in rescue robotics: a scoping review

Linda Battistuzzi, Carmine Tommaso Recchiuto, Antonio Sgorbissa

Sam Trowbridge

As the field of Robotics continues to grow and increasingly efficient autonomous machines are developed, one concern often brought to the forefront is the ethical implementation of these Machines. This paper reviews these ethical issues facing autonomous search & rescue robots. These machines, capable of saving human lives, face some of the most sensitive ethical issues much like autonomous vehicles because of the exact task they have been designed to enact.

The common academic ethical concerns of autonomous machines consist of: discrimination, reliability and safety, however trust and expectations of users and possible rescuee’s are just as vital. The “expectation gap” of robots as discussed by Linda Battistuzzi et al can lead to misconceptions of a machine's capabilities. As autonomous technology is increasingly integrated with everyday tasks, the understanding of how this technology functions is rarely known by the average person.

This can lead to overestimation of the machine's intelligence, also through incorrect usage of a machine underestimation. Expectation and understanding directly correlates with the level of trust an individual considers towards autonomous machines. In a dangerous scenario where search & rescue robots are utilised, trust and expectation could lead to failure and possibly the loss of human life. Therefore it is vital that users fully understand the functionality of these machines and their capabilities.

### 8.4 Logic-Based Technologies for Intelligent Systems: State of the Art and Perspectives.

Calegari R, Ciatto G, Denti E and Omicini A (2020)

Jack Belham

Sub-symbolic AI techniques such as Machine Learning aim to make predictions and utilise big data to make autonomous decisions. A lot of these systems, such as those with neural-based architectures lack explainability of the decisions that are being made. This explainability is present in symbolic AI techniques such as logic-based systems. Within the paper, the idea of eXplainable AI (XAI) is explored where symbolic AI techniques are used to explain sub-symbolic AI techniques. This is important in building trust, interpretability and observability especially in AI systems being utilised in legal and medical settings.

Whilst there was research into the possibility of extracting symbolic knowledge from trained numeric neural-architecture models in the 80s and 90s, little research was conducted within this area at the time. More recently, fuzzy neural networks, which combine neural networks and fuzzy logic together, are being used to provide a balance of between accuracy and interpretability. These models combine human-like reasoning with the learning approach of neural networks. Here, logic based systems are used to address the issue within XAI. This is currently a very active research area as of right now. However, an overall framework for such a system has still not been developed.

# 9. Conclusion

Within this paper, we have analysed different heuristic methods for attempts, hits and ship placements within the game of gunships. The heuristic agent was found to perform a lot better than a random agent, even being able to exploit a “misses” game mechanic. An improvement that could have been made would have been to calculate the probabilities assigned to each cell for the probability of a hit. This could then have then replaced the hit and attempt heuristic. This would likely have yielded better results, although at the expense of computational time required for each move.

Minimax was applied to the game under the assumption of perfect knowledge of the board state. When applying iterative deepening to calculating the next attempt of a game, this yielded perfect game results up to a depth of 15. This depth was achieved through using the simplifying assumption that any other cell besides a blue hit or red hit yielded a score of 0. Zobrist hashing was originally considered to store the board states, but was not implemented due to a lack of time. This could help reduce the time taken to recalculate the heuristic of which player was winning when revisiting the same board state again.

Predicate logic was applied to the game of gunboats. A knowledge base was defined for the game, and a particular board instance was considered. Using resolution on this board state, we proved that a not tried cell must be a hit. We could have considered more different board states to ensure that the knowledge base holds for every situation within gunboats.

We considered the game using probabilistic reasoning by modifying what we had done for the heuristic in order to consider situations where misses do not stay as misses. This required considering all of the possible ship placements for the opponent given the current board state. This is computationally expensive where often many thousands of board states were considered in order to calculate the probability of each square being a hit. Monte Carlo sampling could have been used to reduce the number of board states that needed to be considered in order to give a good representation of the probabilities of a hit.

Overall, we have conducted a thorough examination of our implemented algorithms and have discussed the results in detail. We have suggested various improvements that could have been made at each stage, and considered some of the ethical and legal issues that surround the search methods and logic reasoning that we have implemented.

# References

Calegari R, Ciatto G, Denti E and Omicini A (2020) Logic-Based Technologies for Intelligent Systems: State of the Art and Perspectives. *Information*. Information 11(3): 167.

Linda Battistuzzi, Carmine Tommaso Recchiuto, Antonio Sgorbissa “Ethical concerns in rescue robotics: a scoping review” Ethics and Information Technology (2021) Volume23 pp 863–875. URL: <https://link.springer.com/content/pdf/10.1007/s10676-021-09603-0.pdf>

Mittelstadt, B., Allo, P., Taddeo, M., Wachter, S. and Floridi, L., 2016. The ethics of algorithms: Mapping the debate. *Big Data & Society*, 3(2), p.205395171667967.

Soares, N. and Fallenstein, B. (2014) Aligning superintelligence with human interests: A technical research agenda. Machine Intelligence Research Institute (MIRI) technical report 8. URL: <http://intelligence.org/files/TechnicalAgenda.pdf>

Vamplew, P., Dazeley, R., Foale, C., Firmin, S. and Mummery, J. (2017) “Human-Aligned Artificial Intelligence is a Multiobjective Problem”, *Ethics and information technology,* 20(1), pp. 27-40. doi: <https://doi.org/10.1007/s10676-017-9440-6>