GRP 26: Battleship

Frankie Cao 20341005

Christine Ma 20340949

Jacob Roe 20351389

Nicholas Wang 20364023

Course Modelling Project

CISC/CMPE 204

Logic for Computing Science

December 8, 2023

Abstract

This project is designed to evaluate the best possible guess at any given stage of a game of Battleship. Our model will interpret the known hits and misses from previous turns, and apply our understanding of probabilities to suggest the optimal next move.

Propositions

Maps:

Each player has two maps in battleship, those being the Primary, and Tracking maps, Both of which will be formatted as grids, with the x-axis being labeled A-J, and the y-axis being labeled 1-10.
The **primary** is a map that records every attack the opponent makes, and will be denoted as:

$$P_{PlayerNumber}(X - Coord, Y - Coord).$$

The **tracking** is a map that records the attack the player did to the opponent, and will be denoted as:

$$T_{PlayerNumber}(X - Coord, Y - Coord)$$

Guessing:

Guesses are taken on a turn based system, with each player guessing a coordinate to attack, once a player has guessed a coordinate, the apposing player will then respond with whether the guess was a hit or miss, evaluating to TRUE or FALSE respectively.
Guesses will be denoted as:

$$G(GuessNumber)_{PlayerNumber}(X - Coord, Y - Coord)$$

And will affect the current players Tracking Map, and the apposing players Primary Map based on the result of the guess.

Ships:

Each player has five ships that they may place before the guessing phase of the game takes place, these ships along with their respective lengths and letter notation are: **Carrier 5 (C), Battleship 4 (B), Destroyer 3 (D), Submarine 3 (S), and Patrol Boat 2 (P).**
Ships will be denoted as:

$$S_{PlayerNumber,ShipLetter}(X - Coord, Y - Coord)$$

the X-Coord and Y - Coord will hold the location of each part of the ship per ship type.

Constraints

List of constraint types used in the model and their (English) interpretation.

Placement Constraints:

Ships must be placed so that all component coordinates of the ship are on the predefined map:

$$S_{P,L}(x,y)$$
$$0 \leq x,y \leq 10$$

Ships may not be placed touching one another:

$$S_{P,L} \rightarrow \neg(S_{X,Y} \wedge ... \wedge S_{A,B})$$

Gameplay Constraints:

The game ends once all ships of either of the two players are sunk:
This constraint will be denoted as:

A ship is destroyed once all of its component coordinates are hit where (a,b) is the starting coords of the ship and, (x,y) are the ending coords of the ship:
This constraint will be denoted as:

$$S_{P,L(a,b)} \wedge ... \wedge S_{P,L(x,y)} \rightarrow S_{P,L}$$

When a ship is destroyed, the coordinates around that ship cannot contain a ship and are "revealed" to the attacking player. An example for a horizontally orientated ship:

$$S_{P,L} \rightarrow \neg(S_{P,L(a-1,b+1)} \wedge ... \wedge S_{P,L(x+1,y+1)}) \wedge \neg(S_{P,L(a-1,b-1)} \wedge ... \wedge S_{P,L(x+1,y-1)}) \wedge \neg(S_{P,L(a-1,b)}) \wedge \neg(S_{P,L(x+1,y)})$$

Guessing Constraints:

A player may guess any coordinate defined within the confines of the map:

$$(G_P(A,1) \vee G_P(A,2) \vee \ldots \vee G_P(A,10)) \vee (G_P(B,1) \vee G_P(B,2) \vee \ldots \vee G_P(B,10)) \vee \ldots \vee (G_P(J,1) \vee G_P(J,2) \vee \ldots \vee G_P(J,10))$$

If a new ship is hit for the first time, the next guess should be in a cardinal direction relative to that guess:

$$G(n)_P(x,y) \wedge \neg S_L \rightarrow G(n+1)_P(x+1,y) \vee G(n+1)_P(x-1,y) \vee G(n+1)_P(x,y+1) \vee G(n+1)_P(x,y-1))$$

$\neg S_L$ being parts of the ship that have not been hit yet

Once a ship is hit twice in some given direction, it can be assumes the ship is orientated in that direction, and the following guesses should be be consistent with that logic:

$$G(n-1)_P(x,y) \wedge G(n)_P(x+1,y) \wedge \neg S_L \rightarrow G(n+1)_P(x+2,y) \vee G(n+1)_P(x-1,y)$$

Model Exploration

We contemplated on how the game board would be implemented, ideas are as followed:

- Pool of pre-made boards (different board variations with ship already placed)
- Idea above + pre-made scores (different variations of hits and misses)
- Smaller board (smaller board size and less boats)
- Some combo of the above
- Original 10 x 10

We figured out that a naive approach of putting every possible setup (legal or illegal) in the original 10 x 10 game with a 5 length, 4 length, two 3 length and 2 length ship, would result in 77,414,400,000 combinations ($120 * 140 * 160 * 160 * 180$).

We concluded that we should start with a smaller board using only a 5 length, 4 length and 3 length ship.

Two different models were created to explore the different ways it could be implemented.

In the early stages one member's implementation in a 6 x 6 board showed them the number of games where a ship in each coordinate is (with the placement constraints in consideration):

$$\begin{bmatrix} 1356 & 2748 & 3528 & 3552 & 2796 & 1404 \\ 571 & 1155 & 1487 & 1479 & 1139 & 556 \\ 936 & 1888 & 2416 & 2416 & 1888 & 936 \\ 936 & 1888 & 2416 & 2416 & 1888 & 936 \\ 556 & 1140 & 1480 & 1487 & 1155 & 571 \\ 1403 & 2795 & 3551 & 3527 & 2748 & 1356 \end{bmatrix}$$

The member noticed a discrepancy, indicating that common sense would make it so the middle would have the highest chance. Considering ships can not touch, the member defined a proposition to find every game that has boats that are touching and considers them non existent. Including all ship being used on the 6 x 6 board, every possible configuration is approximately 41472. We also we wondering why the result were not symmetrical or close to symmetrical and deliberated that it could possibly be because of all the different variations

The member had doubts about defining the boats, so another member revised it, but the result was similar to the previous:

$$\begin{bmatrix} 1248 & 2496 & 3168 & 3168 & 2496 & 1248 \\ 416 & 832 & 1056 & 1056 & 832 & 416 \\ 832 & 1664 & 2112 & 2112 & 1664 & 832 \\ 832 & 1664 & 2112 & 2112 & 1664 & 832 \\ 416 & 832 & 1056 & 1056 & 832 & 416 \\ 1248 & 2496 & 3168 & 3168 & 2496 & 1248 \end{bmatrix}$$

We conclude that there was not a discrepancy and it was meant to be that way.

The other implementation was inspired by the previous implementation, along with a specific student's feedback. The feedback had the idea that running one large model could quickly exceed the amount of resources to run the solver, so solution was to break the model down into individual turns. The next steps to implement this is:

- Generating New Guesses Based on the Frequency Map:
  Next guess is the highest score on the frequency map because it is the most probable to contain a ship.
- Handling Hits and Misses:
  Hit : next guess considers adjacent coordinate
  Miss: Frequency map will update to not consider that coordinate
- Continuing Until End:
  iteration based on results, recalculating frequency map and making new guesses based on new map. Continuing until all ships are hit.

After the frequency map updates we could then use some constraints to help guide the SAT-Solver to what was the most optimal move. For instance, if a guess resulted in a hit, then that would imply that there is a boat in a cardinal direction of that square. While a human would need to refine this idea by looking at what size are the remaining boats which are not destroyed to get a better sense of possible orientations, the frequency map would tell the model which direction would result in a more likely outcome. Furthermore, since boats cannot be placed around each other, when a boat has been destroyed we could also add constraints that imply in this game there are no boats in the surrounding area of the destroyed boat. Hence, it would be meaningless (or illegal, depending on implmentation) to guess in one of those squares.

Problems and Workarounds

When building the model for our project initially we planned on making a large model and having the SAT-Solver break down the model and give us all the possible games. However, when doing this since there are so many different combination even with a smaller scale problem the compilation time was too long and thus unsustainable.

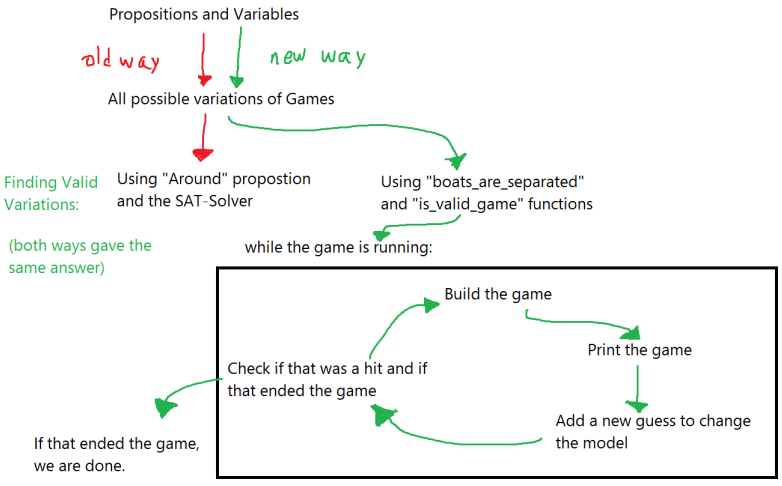
So, we decided to just use regular logic to add the games into a list where we could then use the SAT-Solver to answer questions about our model such as

finding the most optimal move.

When we rebuilt the game after added a guess, we encountered an issue where there were no more solutions. After analyzing our implementation we believe this was due to the Encoding object having the Game proposition for the "valid" (i.e. possible) games. So, after the hit or miss it would also add the constraint that a Game was false. However, since the encoding object had the constraint that that game was also true this resulted in a contradiction which broke our model.

Our initial solution was to use the included E.clear_constraints() in the library documentation. However, this resulted in no change. We also tried implementing new propositions named Miss and Hit, however, these came with their own issues and bugs so we opted out of this idea. After struggling for some while we referenced the peer feedback and one of the students mentioned that there was a function called E._custom_constraints.clear() and this function worked for our purposes so we used it to clear the constraints and build a new theory for each iteration of the game.

In theory, the locations of the boats should be symmetrical horizontally and vertically, this most likely indicates an issue with our model. However, if the model is correct this means we do not understand the problem. There are thousands of games so unlike a smaller scale project like Professor Muise's Graph Properties project we are unable to debug this simply by enumeration. Below, we have a diagram to indicate how our model deviated from the original implementation and how the new one should work.



Another Issue with our model is despite painstaking effort, the frequency map of where the possible boats could be now updates however the values in

the map are most likely incorrect. For instance, when a guess is selected at a certain coordinate and it results in a miss, this should mean the number of possible games with a boat at that coordinate equals zero. Similarly, if that guess resulted in a hit, all possible games with their boats not on that coordinate should be removed. However, in our model misses do not reduce the number down to zero, however it does significantly reduce the number of games where a boat is in that spot. Unfortunately because of this roadblock, we were not able to implement all the mechanics we had in mind such as a game end system, and the revealing of the area around a destroyed ship to filter out more variations.

First-Order Extension

Updated Propositions (Predicates):

P(x, y, z) : x and y are the coordinates of hits / misses made by opponent, z indicates player

T(x, y, z) : x and y are the coordinates of hits / misses made by player, z indicates player

G(x, y , z) : x and y are the coordinates guessed, z indicates player

S(w, x, y , z) : w is the type of ship, x and y are the coordinates ship touches, z indicates player

Updated Constraints:

$\forall w, x, y.S(w, x, y, z)$: End game requirement

$\forall x, y.S(w, x, y, z)$: Coordinates needed to hit to destroy ship

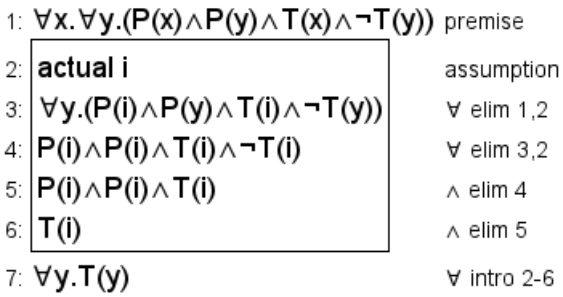
$\exists x, y.G(x, y, z)$: Guesses of coordinates defined by map

A good first-order extension would allow us to reduce the number of variables we would have to iterate over to get a working model of our theory. For instance, instead of having to look at the individual coordinates in every ship we can use syntax that describes ideas such as "for all coordinate in a ship". This is a powerful concept that can be used to greatly reduce the run time of very large models such as Battleship. For our first-order extension we postulated the new set of propositions and constraints would look something like the above.

Jape Proof Notes

Proof 1: A player can only make a guess when it is their turn

Let P(x) be "x is a player" and let T(x) be "x has taken their turn."

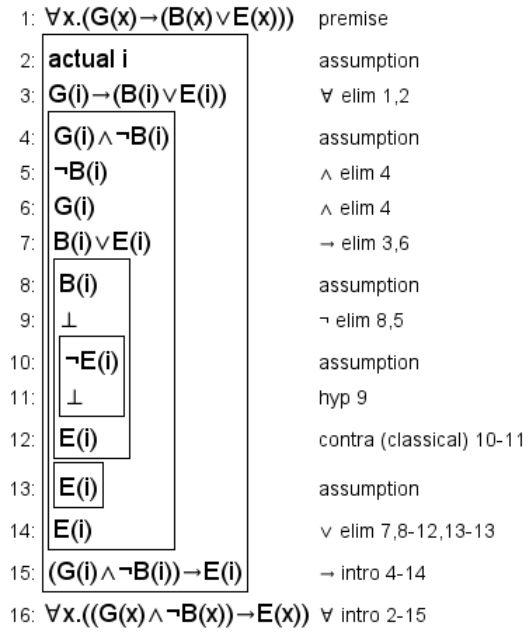


For all players x and y, if x and y are different players, and player x's turn but not player y's turn, then player y must take their turn.

This ensures that a player can only go if it is their turn.

Proof 2: If a guess is not on a ship, it is a miss

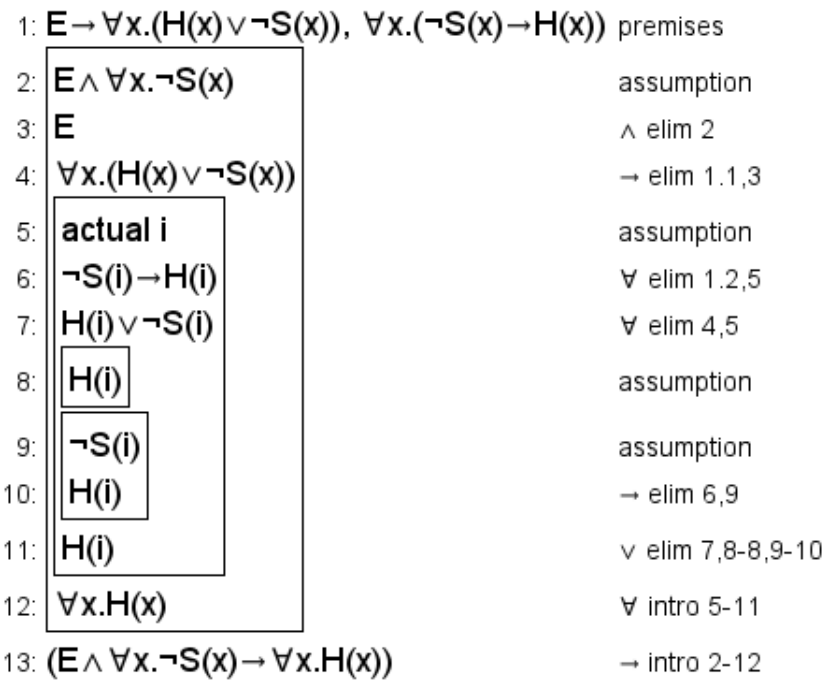
Let G(x) be "Spot x is guessed." , B(x) be "There is a ship at spot x." and E(x) be "Spot x is empty/a miss."



When guessing there is either a ship or a miss at that coordinate. If there is no boat, then it is a miss.

Proof 3: If all boats are sunk, game ends

Let S(x) represent a ship, H(x) represent a ship has been sunk and E represent game end



Game ends when coordinates don't have a ship or a ship has sunk on that coordinate, where no ship can also mean ship has sunk, such that when the game ends and the there no ships on any coordinate, then it means all ships have sunk.