# **BATTLESHIP (DRAFT)**

PROJECT SUMMARY

Battleship is a two player game where each player tries to sink all five of the enemy’s hidden ships on a grid. The first player to sink all of the enemy’s ships wins. However, we introduced a unique twist by modifying the traditional ruleset.

In this version, the board will be pre-made, with markers designating each position as either water, ship, or miss. Based on this predefined layout and a set of propositions and constraints, the computer will calculate potential locations for ship segments.

# PROPOSITIONS

* Hit(location): Specifies that a particular location on the board has been hit.
* Boundary(location): Specifies that a particular location is a boundary (ship segment cannot be on a boundary).
* Ship(location, stype): Specifies that a ship of a particular type (e.g., currently a destroyer or submarine) is at the given location.
* PossibleSegment(location): Added by the computer to decide where ship segments are possibly located

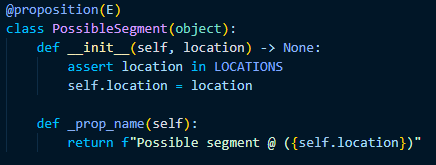
# CONSTRAINTS

* No hit and miss in the same spot. The location cannot be marked as a hit if it is a miss, also there cannot be a possible segment on a missed spot, we have,
  + ¬(Hit(x, y​) ∧ Miss(x, y))
  + ¬(PossibleSegment(x, y) ∧ Miss(x, y))
* For any spot that is hit, there cannot be a possible segment on that spot. Also, there are possible segment surrounding that hit location, so:
  + ¬(Hit(x, y) ∧ PossibleSegment(x, y))
  + Hit(x, y) ⟶ PossibleSegment(x+1, y) ∧ PossibleSegment(x, y+1) ∧ PossibleSegment(x-1, y) ∧ PossibleSegment(x, y-1)
* Boundary locations cannot be hit, we have,
  + ¬(Boundary(x, y) ∧ Hit(x, y))
* Assigning Ship Types: If there are 3 consecutive hits in a row horizontally, they must belong to a destroyer ship, 2 consecutive hits means a submarine, we have,

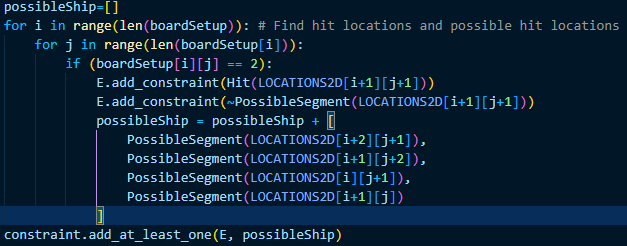
# MODEL EXPLORATION

## Addition of the PossibleSegment proposition

We were originally just going to use hit/miss/ship statuses for each location on the board but we realised that PossibleSegment was necessary for the computer to “decide” where ship segments are located.

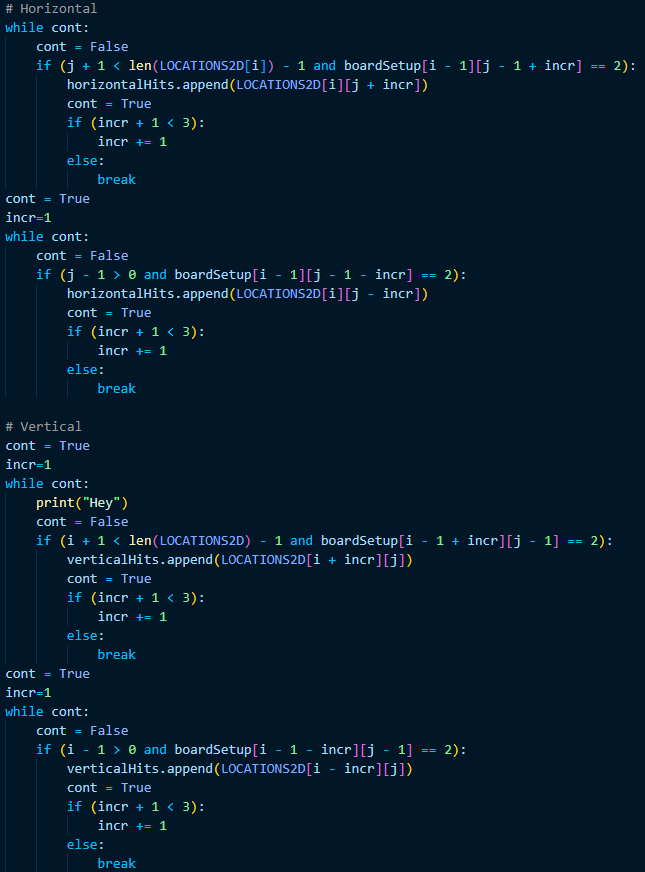


While Hit is used for showing the current state on the board, PossibleSegment is different in that it’s the way that the model decides what to do next.



Implementing the FindShipType() function

As we worked through implementing the model, we realised that we needed some way to distinguish between ships on the board. This led to us creating the findShipType function. This quickly became much more complicated than we expected which led us to using many while loops and if statements to decide the type of ship instead of logic methods.



At the end of the function a simple constraint is set including the location and type. The function works correctly for simple cases, but for something more complex like ships in a “T” shape, it fails to differentiate between them. We also realised just how inefficient this code is, and it will be improved later.

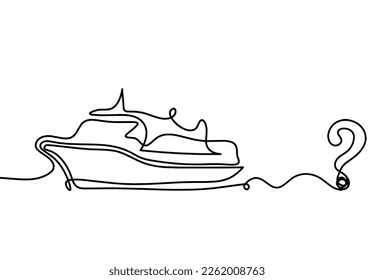
## 

# JAPE PROOF IDEAS

Nothing for now, would like advice on where to start.

# REQUESTED FEEDBACK

1. Looking at the constraints and propositions, what adjustments or additions would you recommend to improve?
2. We’d like for our model to be able to “have multiple turns” but with our current setup that seems like it will be difficult. Do you have any advice that would allow us to use previous constraints to allow the model to guess ship locations (eg. if there is only one solution for possible segments, then the computer “places” a ship there and if there’s more than 1 it guesses a random location)?
3. How should we begin our Jape proofs?



# FIRST-ORDER EXTENSION

PENDING. Not started as of yet.