# **BattleShip**

# 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. We will be using this ruleset for the model: [Battleship](https://www.hasbro.com/common/instruct/battleship.pdf)

Our team proposes to create a model of battleship to solve for a winning scenario. This will be done by determining the possible ship segment locations with the propositions and constraints given. The project's goal is to blend classic game development with formal logic, providing both an entertaining experience and a practical application of logical reasoning in programming.

# Propositions

TODO

* Bring in the figure for the edge labels
* Make sure they are numbers and not letters
* Finish rewriting to mirror the names used in the code

Draft:

* Hit(location): Specifies that a particular location on the board has been hit.
* Boundary(location): Specifies that a particular location is a boundary and cannot have a ship.
* Ship(location, stype): Specifies that a ship of a particular type (e.g., destroyer or submarine) is at the given location.
* TileConnection(t,x,y): says that edge point x is connected to edge point y, on tile t. There are 7 possible target points for every individual entry point. E.g., in the figure to the top right for tile T1, we could have edge\_connected(H,C,T1) – this is shown with the red line
* configuration(t, o): tile t is oriented in direction o. The orientation can be NSEW, and we will assume that A,B endpoints are on the N side (again, see the figure above)
* location(t, loc): tile t is at location loc. The location is where on the board that things exist.
* at(p, t, ep): piece p is on tile t, at the edge point ep (e.g., A, B, …).
* can\_reach(p, t, ep): given the path in front of it, piece p can reach tile t’s endpoint ep
* connected(t1, ep1, t2, ep2): there is a direct connection between tile t1’s edge point ep1 and tile t2’s edge point ep2

# Constraints

TODO:

* Go over the proposal constraints and refine for what’s currently done
* Add the new constraints that came into the project since then
  + Just summarize the super simple ones (e.g., no self loops)

Draft:

* No Hit and Miss in the Same Spot. The location L\_{i+1, j+1} cannot be marked as a hit if it is a miss,we have,

¬Hit(Li+1,j+1​)

* ¬(Boundary(Li,j​)∧Hit(Li,j​))
* Boundary Cannot Be a Hit. A boundary cell cannot also be a hit,we have,

¬(Boundary(Li,j​)∧Hit(Li,j​))

* Exactly One Neighboring Hit Constraint. If there is a hit at a specific location, then exactly one of the neighboring locations must also be a hit. For four neighbors n\_1, n\_2, n\_3, n\_4 we have,  
  (edge\_connected(x,y,t) /\ can\_reach(p, t, x))  can\_reach(p, t, y)

(Hit(n1​)∨Hit(n2​)∨Hit(n3​)∨Hit(n4​))∧¬(Hit(n1​)∧Hit(n2​))∧…

* Assigning Ship Types to Hit. If there are 3 consecutive hits in a row horizontally, they must belong to a destroyer ship, we have,

∀l∈HorizontalHits,Ship(l,destroyer)

* I am not yet sure how to force can\_reach(p, t, y) to be false, when there is no way to actually reach it. Halp!!!
* If there is a connection in a tile, then the edge points are connected: edge\_connected(x,y,t)  connected(t, x, t, y)
* Connect neighbouring edge points for tiles next to each other. For loc1 and loc2 that are next to each other,  
  “right/top of loc1” /\ “left/top of loc2”  connected(t1, “right/top”, t2, “left/top”)
* “right/top of loc1” needs to be a logical formula that captures the edge point of the tile that is located at loc1, and needs to take the orientation into account. Not yet sure how to best do this.

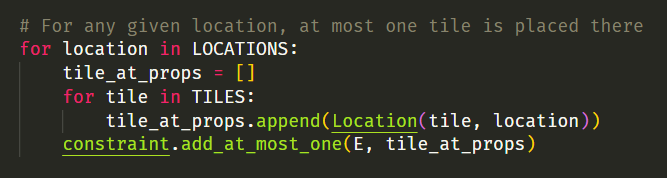
# Model Exploration

*List all the ways that you have explored your model – not only the final version, but intermediate versions as well. See (C3) in the project description for ideas.*

## Fixing the Bug With More Tiles Than Locations

When I started adding more tile possibilities, I found that 5 tiles happened to be placed on only four locations (2x2 grid). This *shouldn’t* be possible. This came about with 5 tiles, and just 4 locations, and looking at the final solution, I found that two separate tiles were assigned to the same location.

This immediately raised the fact that I was missing a constraint saying that at most one tile appears at a location. I fixed this by adding a constraint that looks like:



So for every location, we have this constraint added that says “at most one tile can be there”. Originally, I thought this might have to be “exactly one”, but then decided that I want to allow for some locations to have no tile placed.

After adding this constraint, I confirmed that having 5 tiles and 4 locations was unsolvable.

## Ensuring no Location Connections When No Tile

We wanted to make sure that locations don’t require a tile to be there. But leaving the connections on a location be free to be true/false, means that we might have phantom connections created. To test this theory out, I created an example that I would hope leads to no solution:

## A screen shot of a computer code Description automatically generated

Note that the 3 tiles in the setup are placed at 11, 12, and 21, and that we’re trying to have a location connection on the 22 location. This happened to be solvable, because there was nothing forcing these propositions to be false when no tile was put down. The way I resolved this was to use the following constraints:

A screen shot of a computer program

Description automatically generated

This was also the first time that I had to start using the And and Or functionality. After this constraint, the test case led to an unsolvable theory, just as I had hoped.

# Jape Proof Ideas

*List the ideas you have to build sequents & proofs that relate to your project.*

Haven’t started on my Jape proofs yet, but here are some ideas I might try in the remaining few weeks of the course:

1. For a single tile, if there are links 1-2, 3-4, and not 5-6, then 5 must be connected to 7 or 8.
2. If I can get to a location in 1 step, I can’t get there in 2.
   1. Perhaps, if location distances are unique, then if you get there in 3, you can’t get there in 4
3. Stretch goal: there is no way to get to a location in 2 different distances

# Requested Feedback

*Provide 2-3 questions you’d like the TA’s and other students to comment on.*

# First-Order Extension

*Describe how you might extend your model to a predicate logic setting, including how both the propositions and constraints would be updated.* ***There is no need to implement this extension!***

Nope, just haven’t started this yet.

# Useful Notation

*Feel free to copy/paste the symbols here and remove this section before submitting.*