# Project Summary

In our project, we model the two-player strategy board game, Battleship, and solve a way to cheat, if we were allowed to move some of the ships. A fleet of ships is placed onto a grid. To steer our project in a unique direction, we will explore how players can shift their ships in response to incoming shots to avoid being hit. It is considered a miss if the opponent incorrectly targets a position without a ship, or if the player successfully shifts their ship out of harm’s way. A shot is a hit if the opponent correctly identifies the location of a ship, and the player has no option to move their ship to avoid the hit. Our model uses logical propositions and constraints to represent the game board, ship placements and shifts, and shots fired systematically.

# Propositions

Propositions are used to represent the Battleship model through logical true or false statements. We have categorized our propositions into distinct groups which focus on unique aspects of the game: ship positioning and movements, shot outcomes, sunk ship status, and player turns.

Ship Propositions:

shipY\_xij(t): True if the position (i, j) on the grid is occupied by ship x at time t.

bumpedY\_xij(t): True if ship x has been moved to an adjacent position (bumped) from its original position at time t.

shipY\_movable\_x(t): True if ship xcan be moved to a new position without violating game constraints at time t.

adj(i, j, k, l): True if positions (i, j) and (k, l) are adjacent on the grid.

Shot Propositions:

shotY\_ij(t): True if the position (i, j) has been shot (fired at) by a player at time t.

hitY\_ij(t): True if the position (i, j) was a hit at time t, meaning that a ship occupies the position and cannot be moved to avoid the shot.

missY\_ij(t): True if the position (i, j) was a miss at time t, meaning that the shot was fired at an empty position or the ship successfully shifted away.

Sunk Ship Proposition:

sunkY\_x(t): True if ship xhas been sunk at time t.

Turn Indicators:

turn1(t): True if it is player 1's turn at time t.

turn2(t): True if it is player 2's turn at time t.

# Constraints

1. Ship Placement Constraints:
   * No two ships of the same player can occupy the same cell:
     + (x != y) → (shipY\_xij(t) → ¬shipY\_yij(t))
       - x and y represent different ships for player Y
   * A ship can occupy multiple cells: e.g. If a carrier occupies 3 cells:
     + ship2\_carrierA1(t) ∧ ship2\_carrierA2(t) ∧ ship2\_carrierA3(t)
   * Sunk Ships are removed
     + sunkY\_x(t) → ¬shipY\_xij(t)
   * Positions (i, j) and (k, l) are adjacent if they are directly next to each other horizontally or vertically.
     + adj(i, j, k, l) ↔ (∣I − k∣ + ∣j − l∣ =1)
2. Shot Constraints:
   * A shot is either a hit or a miss, but not both
     + shotY\_ij(t) → (hitY\_ij(t) ↔ ¬missY\_ij(t)
   * A shot is a hit if there is a ship there:
     + Example for B5 and Player 1 shooting at Player 2's carrier:
       - hit1\_B5(t) ↔ (shot1\_B5(t) ∧ carrier2\_B5(t))
   * A shot is a miss if there is no ship there:
     + Example for B5 and Player 1 shooting at Player 2's carrier:
       - miss1\_B5(t) ↔ (shot1\_B5(t) ∧ ¬carrier2\_B5(t)
3. Bumping Mechanism:
   * Allow ships to be bumped to adjacent positions after each opponent shot (or stay):
     + Example for Ship x moving from A1 to A2:
       - bumped1\_x(t) ∧ ship1\_xA1(t−1) → (ship1\_xA2(t) ∧ adj(A1, A2))
   * Only one ship bump per turn is allowed (bump one ship based on the opponent’s shot)
     + turn1(t) → (¬turn2(t) ∧ bumped\_xij(t))
4. Turn-Based Constraints
   * Only the player whose turn it is can make a move
     + Player 1:
       - turn1(t) → (shot1\_ij(t) ∨ bumped1\_xij(t))
     + Player 2:
       - turn2(t) → (shot2\_ij(t) ∨ bumped2\_yij(t))
   * After one player takes their turn, it should be the other player’s turn
     + (turn1(t) → turn2(t+1)) ∧ (turn2(t) → turn1(t+1))
   * If a player has no ships remaining (game is over), neither player can take a turn
     + (game\_over(t) → ¬turn1(t) ∧ ¬turn2(t)
5. Game End Condition
   * The game ends when all ships of one player are sunk.
     + ∀x (sunkY\_x(t)) → game\_over(t)

# Model Exploration: Explanation and Results

We started by modelling a basic Battleship setup using logical propositions and constraints to represent the different parts of the game. This includes setting up a grid as the game setting and using logic to track ship placements, hit-and-miss outcomes, and turn-taking. In this setup, each player places their ships onto the grid and takes turns firing shots, to sink all the other player's ships. Each shot targets a specific cell (coordinate), and the outcome depends on whether that cell is occupied by a ship or not.

As we progressed, we decided to pivot to a new concept: the "Battleship cheats" model. This model would allow players to adjust, or "nudge", their ship positions in response to opponent shots - adding a sort of in-game "cheat". To add strategic depth to this mechanic, we explored two main approaches. We considered allowing the opponent to fire multiple shots per turn while the defending player would reposition their ships using the model to minimize the amount of damage taken. However, to keep it from being too complex, we settled on a one-shot-per-turn format, where the defending player can nudge a ship after each shot to avoid a hit.

To prevent the solution from always being "move the ship out of the shot's path," we decided to add several rules and constraints to add more strategic depth to the cheat mechanic.

Now that we have a working model skeleton and some constraints that encode some aspects of the battleship setting, we can run the code as-is and see if it finds solutions. Looking at what a solution looks likes will give us a baseline understanding of the model’s current state.

As a first step in testing the model, we simply ran it as-is with the current set of propositions and constraints, without introducing any additional complexity or strategic optimizations. This initial run served as a baseline, allowing us to check whether the model was at all satisfiable and to observe the solver’s response to the given conditions. In doing so, we discovered that the model, as initially configured, yielded zero solutions, indicating that the constraints were likely too restrictive or contradictory. This early test gave us the kick in the butt to try and effectively build a satisfiable model:



We relaxed the constraints by using a smaller grid and fewer ships, and picking a known, valid placement for each ship. We omitted some of the complex constraints like bumping and shooting and decided to gradually reintroduce those complex constraints once we confirm the model works!

A computer screen with white text

Description automatically generated

The solver’s output indicates that the model is satisfiable and yields exactly one solution. In this particular solution, Player 1’s turn at time 0 is active while Player 2’s turn is not, and both Player 1 and Player 2’s ships occupy their designated cells without any conflicts. Notably, the variable likelihoods are all reported as 1.00, which suggests that under the single solution found, these propositions are always true (or trivially satisfied due to the absence of constraints). This result provides a baseline confirmation that the model, in its simplified form, can produce a coherent assignment of propositions. It also sets the stage for slowly adding more complex constraints and ensuring that future versions of the model remain satisfiable.

Now let’s update the code to add multiple turns (t=0 and t=1), model a shot, and model sinking ships and game ending. We also define adjacency conditions for the cells in the 2x2 grid.

A computer screen with white text

Description automatically generated

This output confirms that the introduced complexity—multiple turns, adjacency constraints, and the shot-hit-sink sequence—remains logically consistent. The model is satisfiable with exactly one solution, and the solution details match our expectations. Player 1’s ship is placed at (0,0) and Player 2’s ship at (1,1), fulfilling the initial placement constraints. Player 1’s shot at time 0 is successful (hit is true, miss is false), causing Player 2’s ship to be sunk by time 1 (sunk2\_A(1): True), which leads to the game ending at turn 1 (GameEnd(1): True) and ensures no further turns for either player. The adjacency propositions are all true where expected, verifying that the defined grid neighbors are correctly recognized.

A computer screen with white text

Description automatically generated

After updating the code to introduce the bumping mechanism, the updated scenario remains satisfiable, producing multiple solutions that reflect the increased complexity. In this particular solution, Player 2’s ship is both movable and successfully bumped from (1,1) to (1,0), resulting in a hit at the original location. Although this might seem paradoxical, it indicates that the solver can find configurations where bumping occurs while still allowing the shot to count as a hit under the given constraints. With six solutions in total, the model demonstrates that it can accommodate various logical outcomes, maintaining both complexity and satisfiability.

A computer screen with white text

Description automatically generated

What a jump from 6 to 1.5 million! I went back and tried to introduce more than one ship per player, enforcing that no two ships of the same player can occupy the same cell. We continue to use a 2x2 grid and a minimal time horizon. This doesn’t let the solver choose ship placements, but the no-overlap constraints ensure that if at any point the model or future expansions allowed flexible ship placements (hint hint) they would not be overlapped. Our next step is to try adding game end conditions.

A black screen with white text

Description automatically generated

Looks like we figured out what caused the problem in the first place… 😅Hmmm, I’m thinking there are some constraint conflicts. We have to change the way we deal with the timing of when a ship disappears after being hit. Basically, what we have to do is space things out so the ship is hit at one time step and is only considered removed by the next time step. That way we avoid the “ship must be there and not there at the same time” contradiction, allowing the solver to find valid solutions… And just like that:

A computer screen with white text

Description automatically generated

I also updated the variable likelihoods section to use actual game propositions, and changed the grid to a 5x5 while updating the ships to be 2x1s instead of 1x1, making it more realistic and in-scope for our project, yielding this result, wow:

A black and white pixelated background

Description automatically generated

However, for these specific tests they were predetermined moves. Instead of deciding in advance what each player will do each turn, the game now lets players choose their actions based on what's happening in the game. During their turn, a player can pick any spot to shoot at or decide to move one of their ships to a nearby area if the ship hasn't been sunk yet. The game keeps track of where all the ships are, who has been hit, and when a ship is sunk. After each action, again it checks if all of one player's ships are gone, which would end the game.

A screen shot of a computer screen

Description automatically generatedalthough it’s very interesting to check the possible solutions, it’s strenuous to rethink of new variable likelihoods, but I was curious about what the end chances would look like:

A screen shot of a black screen

Description automatically generated  
So, I computed them ^ I guess it makes sense since the model can bump the ship to not lose… But I wanted to investigate further.

A screenshot of a computer screen

Description automatically generated  
this is interesting, isn’t it… I have a feeling some calculations or propositions aren’t right. Upon closer examination, it became evident that both Hit and Miss probabilities were being satisfied simultaneously for each shot, which is logically impossible in the context of the game. This dual satisfaction likely stems from the constraints not adequately enforcing the mutual exclusivity between Hit and Miss outcomes. Specifically, the constraint meant to ensure that a shot results in either a hit or a miss—but not both—appears to be insufficiently stringent. To address this, I revisited the constraint definitions within the model. I introduced an explicit constraint that prohibits both *Hit* and *Miss* from being true at the same time for any given shot. After implementing these corrections, I recomputed the likelihoods, sure it would make sense!

A screenshot of a computer

Description automatically generated

Looking at the new results, I noticed that the game end likelihoods increase steadily with each turn, reaching up to 72% by Turn 4. This suggests that the game is very likely to conclude as more turns are played, which seems reasonable. However, when I examined the ship sunk likelihoods, I saw that both players' ships have varying probabilities of being sunk each turn, with some ships having a high chance of being sunk early on and others maintaining significant probabilities even by the last turn.

While significant progress has been made in defining ship positions, turn-based actions, and game termination conditions, the logical consistency of certain outcomes could be slightly improved, for example, making sure Hit+Miss=1.00. By addressing the identified issues and implementing the recommended improvements, the model can achieve greater accuracy and reliability, providing a robust simulation of Battleship gameplay with cheats!

# First-Order Extension

We can extend our model to first order by introducing predicates and constraints that have a more generalized representation of the game’s mechanics. This includes accounting for the ship’s positions, shots taken, turns, and time taken.

**Predicates:**

1. **Ship Propositions:**

* **ShipY(x, i, j, t):** Represents that ship x occupies position (i, j) at time t.
* **BumpedY(x, i, j, t):** Represents that ship x has been bumped to position (i, j) at time t.
* **MovableY(x, y):** Represents that ship x can be moved without violating constraints at time t.
* **Adj(i, j, k, l):** True if positions (i, j) and (k, l) are adjacent (vertically or horizontally)

1. **Shot Propositions:**

* **ShotY(i, j, t):** Represents that a shot was fired at position (i, j) at time t.
* **HitY(i, j, t):** True if position (i, j) was a hit at time t.
* **MissY(i, j, t):** True if position (i, j) was a miss at time t.

1. **Sunk Ship Proposition:**

* **SunkY(x, t):** Represents that ship x has been sunk at time t.
* **GameOver(t):** True if all ships of a player have sunk at time t.

1. **Turn Indicators:**

* **Turn1(t):** True if it is player 1’s turn at time t.
* **Turn2(t):** True if it is player 2’s turn at time t.

**Constraints (modelled in first-order):**

1. Ship Placement Constraints:
   * No two ships of the same player can occupy the same cell:
     + ∀x1​,x2​,i,j,t(ShipY((x1​,i,j,t) ∧ ShipY(x2​,i,j,t)) → ⊥)
       - x1 and x2 represents two different ships of the same player. Having both ships be on the same coordinates (i, j) at time t causes a contradiction.
   * Positions are adjacent
     + ∀i,j,k,l(Adj(i,j,k,l) ↔ (∣I−k∣ + ∣j−l∣ = 1))
       - The Manhattan distance demonstrates positions that are next to each other either vertically or horizontally: ∣I−k∣ + ∣j−l∣
2. Shot Constraints:
   * A shot is a hit or a miss but not both:
     + ∀i,j,t(ShotY(i,j,t) → (HitY(i,j,t) ↔ ¬MissY(i,j,t))
       - Mutual exclusivity between Hit and Miss at the same position (i, j) at time t.
   * A shot is a hit if there is a ship there:
     + ∀i,j,t(HitY(i,j,t) ↔ (ShotY(i,j,t) ∧ ∃x(ShipY x,i,j,t))))
       - Note that Y’ represents opponent’s ship
       - A shot only results in a hit if there is a ship at that chosen location (i,j) at time t
   * A shot is a miss if there is no ship there:
     + ∀i,j,t(MissY(i,j,t) ↔ (ShotY(i,j,t) ∧ ¬∃x(ShipY′(x,i,j,t))))
       - The shot results in a miss at the chosen location if there is no enemy ship there.
3. Bumping Mechanism:
   * Ships can be bumped to adjacent positions after each opponent’s shot:
     + ∀x,i,j,k,l,t((BumpedY(x,t) ∧ ShipY(x,i,j,t−1)) → (ShipY(x,k,l,t) ∧ Adj(i,j,k,l)))
       - If a ship x was bumped at time t, and was at position (i, j) in the previous turn (t-1), it now will occupy an adjacent position (k, l).
4. Turn-Based Constraints
   * Only the player whose turn it is can make a move. They can either fire a shot or bump their ship. This is the case for both Player 1 and Player 2.
     + ∀t(Turn1(t) → (∃i,j(ShotY(i,j,t)) ∨ ∃x(BumpedY(x,t))))
     + ∀t(Turn2(t) → (∃i,j(ShotY(i,j,t)) ∨ ∃x(BumpedY(x,t))))
   * Turns alternate between players
     + ∀t((Turn1(t) → Turn2(t+1)) ∧ (Turn2(t) → Turn1(t+1)))
   * No turns when game is over
     + ∀t(GameOver(t) → (¬Turn1(t) ∧¬Turn2(t)))
       - Once the game ends, neither player is allowed to take any more turns.
5. Game End Condition
   * The game ends when all ships of one player are sunk.
     + ∀Y,t((∀x(SunkY(x,t))) → GameOver(t))
       - If all ships of a player Y is sunk, then the game ends at time t.

# Jape Proofs

No two ships of the same player can occupy the same cell:

A black and white text

Description automatically generated

* ∀x.(S(x) → ¬S2(x)), ∃x.(S(x) ∧ S2(x))) ⊢ ⊥
  + S(x): Ship at position x
  + S2(x): A second ship on position x
  + For every position x, if S(x) is present then S2(x) must not be true at that same location
  + There exists a position x where both S(x) and S2(x) are placed together
  + These two premises results in a contradiction because it is not possible for two ships to be at the same position

Ships can be bumped to adjacent positions after an opponent’s shot:

A black and white text

Description automatically generated

* ∀x.(S(x) ∧ P(x) → (B(x) ∧ A(x))), ∃y.(S(y) ∧ P(y)) ⊢ ∃z.(B(z) ∧ A(z))
  + S(x): A ship at position x
  + P(x): A shot fired at position x
  + A(x): The ship moves to an adjacent position from x
  + For all positions x, if a ship is present and a shot is fired, then the ship is bumped and moved to an adjacent position
  + There exists at least a position y where both a ship is present and a shot is fired
  + This concludes that the ship will be bumped (B) and end up at an adjacent cell (A)

In each player’s turn, they can either shoot or bump:

A black text with black letters

Description automatically generated with medium confidence

* actual i, T(i), ∀x.(T(x) → (S(x) ∨ B(x))) ⊢ S(i) ∨ B(i)
  + T(i): Specific instance of a player’s turn
  + For every player, if it is their turn, they can either shoot or bump.
  + This concludes that for the player’s turn, they can either shoot a ship or bump their ship.