Adversarial Lower Bounds

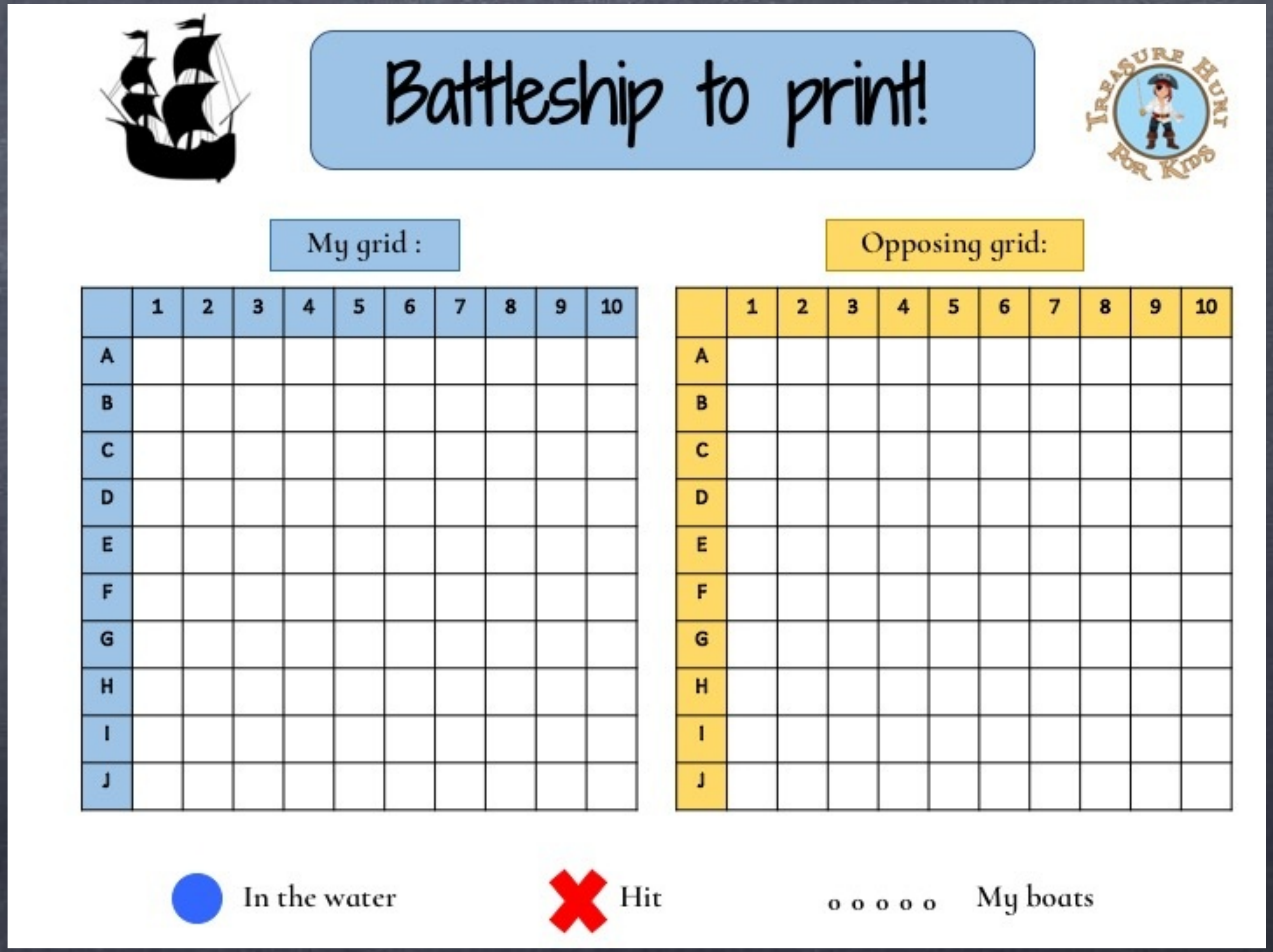
Andreas Klappenecker

Motivation

Battleship Game

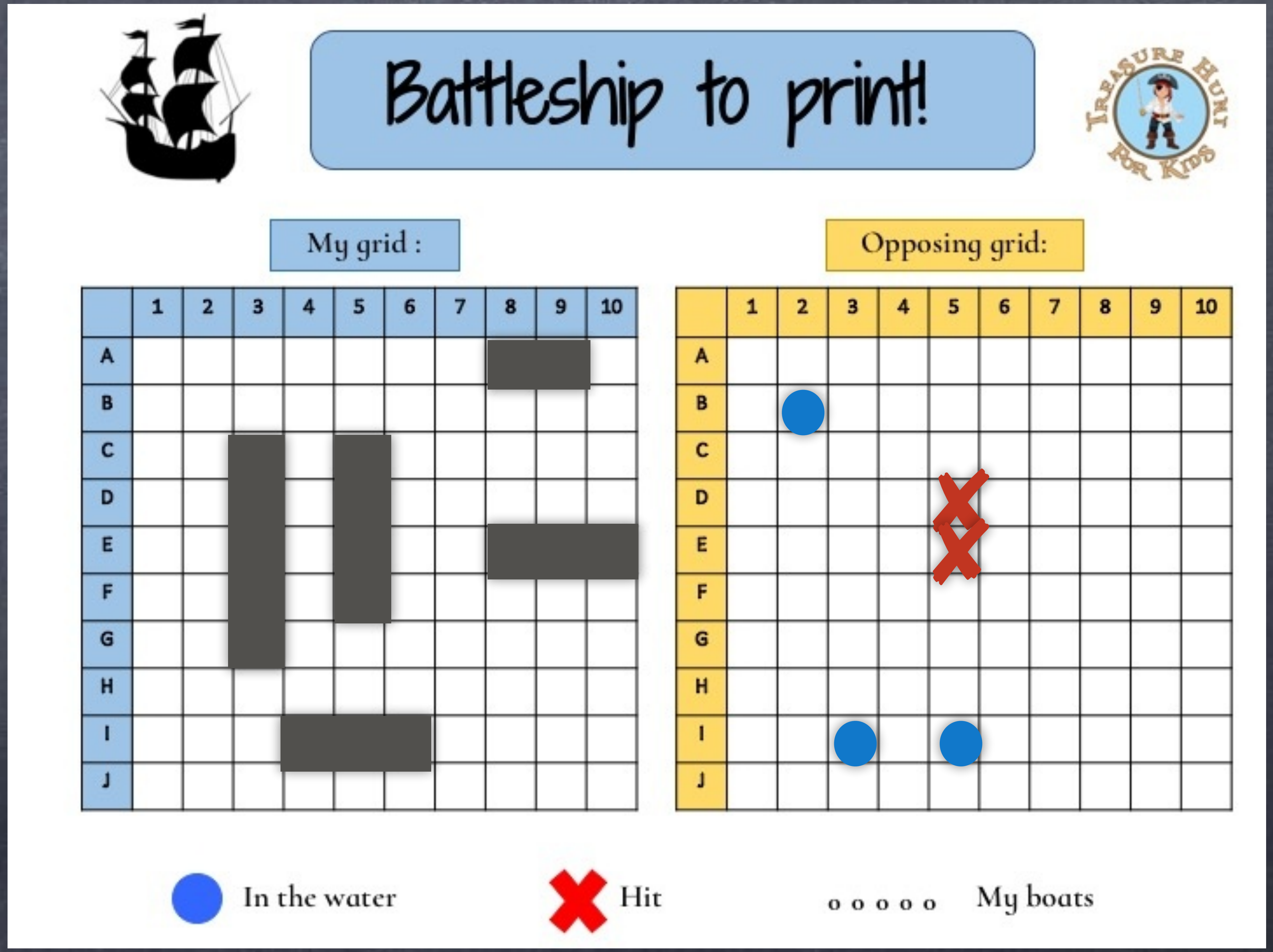
Fleet and Gameboard

- Carrier size 5x1
- Battleship size 4x1
- Destroyer size 3x1
- Submarine size 3x1
- Patrol boat size 2x1



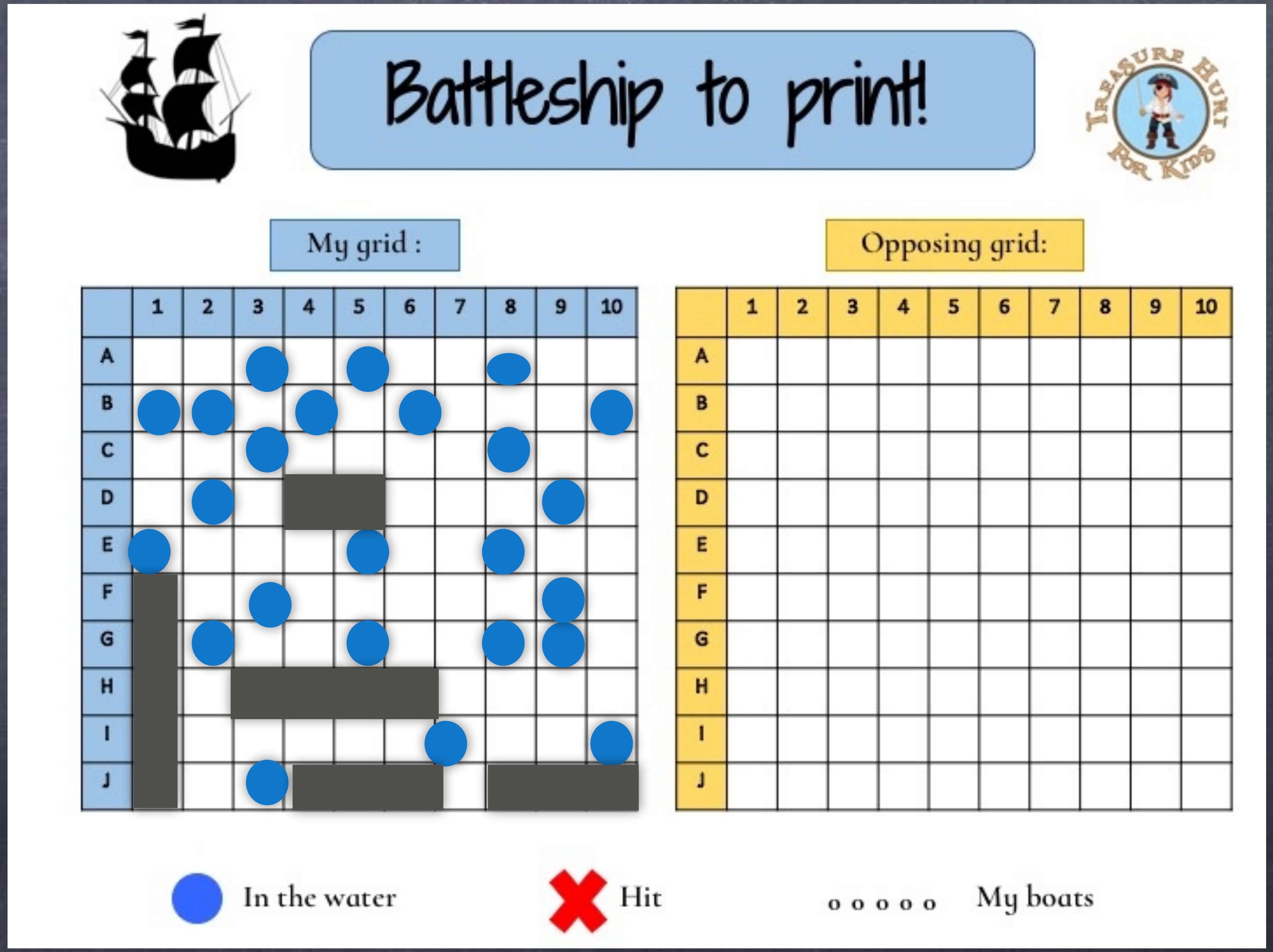
Rules

- Players place fleet
- Players take turns
- In each turn, a player makes a single shot (specifying e.g. B2)
- The opposing player answers whether it was a hit or miss



Bending the Rules

- Suppose that you play now over the phone
- Instead of placing your fleet, you simply answer hit or miss
- You try to drag out the game, so you mostly answer miss
- You keep a record of your answers
- Later you place your fleet so that it is consistent with your answers



The Adversary Technique

The Adversary Technique

Deriving lower bounds for all algorithms that solve a certain problem can be a difficult task.

One potential way to obtain such a proof is by pretending that there is an "adversary" which observes the interaction of the algorithm with its data structures and dynamically generates the input such that the algorithm makes as little progress as possible.

Analyzing the adversary then yields the lower bound proof.

Determining the Minimum

Every comparison-based algorithm for determining the minimum of a set of n elements must use at least n/2 comparisons.

Indeed, every element must be compared at least once, for otherwise the adversary can choose an element that is not compared and set it to the minimum. Precisely two elements are compared by a comparison. Hence, there must be at least n/2 comparisons.

Remarks

- The best comparison-based algorithm know to me makes n-1 comparisons.
- There is quite a gap between n/2 and n-1 comparisons.
- Is the algorithm suboptimal or is the lower bound too weak?

Determining the Minimum

Every comparison-based algorithm for determining the minimum of a set of n elements must use at least n-1 comparisons.

Let's think in terms of a tournament, where x > y means that x won against y. We can declare an element m to be a minimum if and only if every other element has won a comparison against some other element. [As the adversary could declare any element that has never won to be a minimum.] Since each comparison yields one winner, there must be n-1 comparisons.

Lower Bound for Sorting

[Adversary Version]

Any comparison-based sorting algorithm needs in the worst case $\Omega(n \log n)$ comparisons to sort n elements.

The sorting algorithm needs to decide between n! different permutations of the input data. The adversary maintains a list L of all possible input data that are consistent with the comparisons that have been made so far.

Lower Bound for Sorting 2

Suppose the algorithm makes the comparison $a[i]<a[j]$. The adversary computes:

List L_y of permutations in L such that $a[i]<a[j]$.
List L_n of permutations in L such that $a[i]>=a[j]$.

The adversary returns “true” and sets $L = L_y$ when $|L_y| > |L_n|$; otherwise “false” and $L = L_n$.

So the adversary tries to keep the list L as large as possible.

Lower Bound for Sorting 3

Since an algorithm cannot terminate unless $|L|=1$, one needs at least $\log(n!)$ comparisons.

As $\exp(x) > x^n/n!$ holds for all $x>=0$, we have $e^n > n^n/n!$ and hence $n! > (n/e)^n$.

Therefore, we need
$\log(n!) > \log((n/e)^n) = n \log n - n \log e = \Omega(n \log n)$
comparisons.