Battleship Probability Calculator

Project Description

Kendrick Cancio (kdc57)
Saloni Joshi (lsj38)
Angeline Pinilla (ang58)

Overview

Battleship is a two-player competitive game in which each player commands a fleet of battleships located on a grid. Players take turns calling shots into each other's grid in an effort to hit each other's ships and sink them once all components have been hit. Play continues until a player has sunk all of the opposing player's ships. In this project, we aimed to create a probability calculator to assist the player in locating these ships.

Background

In battleship, each player commands a total of 5 ships of varying lengths.
These are:

- Carrier - 5 units
- Battleship - 4 units
- Cruiser - 3 units
- Submarine - 3 units
- Destroyer - 2 units

At the start of the game, both players place their ships on a 10x10 grid so that the ships are:

1) Entirely contained in the board
2) Not overlapping with each other on any grid spaces
3) Placed horizontally or vertically

Play then proceeds to the "shooting" phase where players take turns shooting at each other's boards in an effort to find and sink each other's ships. This is where our battleship calculator comes into play. Typically, players will shoot randomly into an opponent's board using intuition at best: shooting into generally open areas without considering the probability of finding a ship there. Ideally the player would be able to leverage the information gained from all their previously fired shots to maximize their current chances of hitting a ship.

Assuming that an opponent places their ships randomly on their side of the board, we can calculate the true probability of hitting a ship for every cell if we consider all of the

possible ship arrangements given the remaining ships to be found and the current state of the board (all the shots) fired.

Data

In our project, we generate all of our data using our code and user input. The user indicates the current state of their board (locations of all the shots they've fired) and the ships that still remain to be found. Our code then calculates the probabilities of hitting a ship for each cell. *Note* Our model assumes you are searching for ships to sink (not actively trying to sink one you've already discovered) so "hits" and "misses" are not distinguished separately in the user input.
One difficulty in actually calculating the probabilities for each cell is that the number of possible combinations of ship configurations is extremely large, in fact, too large to compute explicitly. To work around this, we simply estimate the probability by sampling a large number of ship configurations and tallying, for each cell, how many of those configurations result in a ship segment being placed on that cell.
Achieving this is a bit trickier than it seems as we must generate samples that not only meet the game criteria for ship placement but also don't conflict with user input. The code works as follows:

For each ship left to be found:
    Generate and save a list of valid locations and orientations where that ship can be placed given the cells indicated as "shot" in the the user input

For a maximum of M iterations or N samples (whichever comes first):
    For each ship left to be found:
        Randomly choose one of the valid locations for this ship
    Given the chosen valid location for each ship, if none of them overlap/conflict:
        Count this as a valid sample and tally all the cells containing a ship segment
    Count this as an iteration

Return the number of samples taken and a matrix of the number of times a ship segment was found in each cell

In addition to calculating the probabilities, because we are taking a sample rather than calculating the true value, we chose to calculate a confidence interval for the statistic so that we could give the user the ability to see how reliable the result is. We use the true confidence interval for a binomial random variable which is defined as:

P_{UB} = 1 - BetaInv(a/2, n-k, k+1)
P_{LB} = 1 - BetaInv(1-a/2, n-k+1, k)

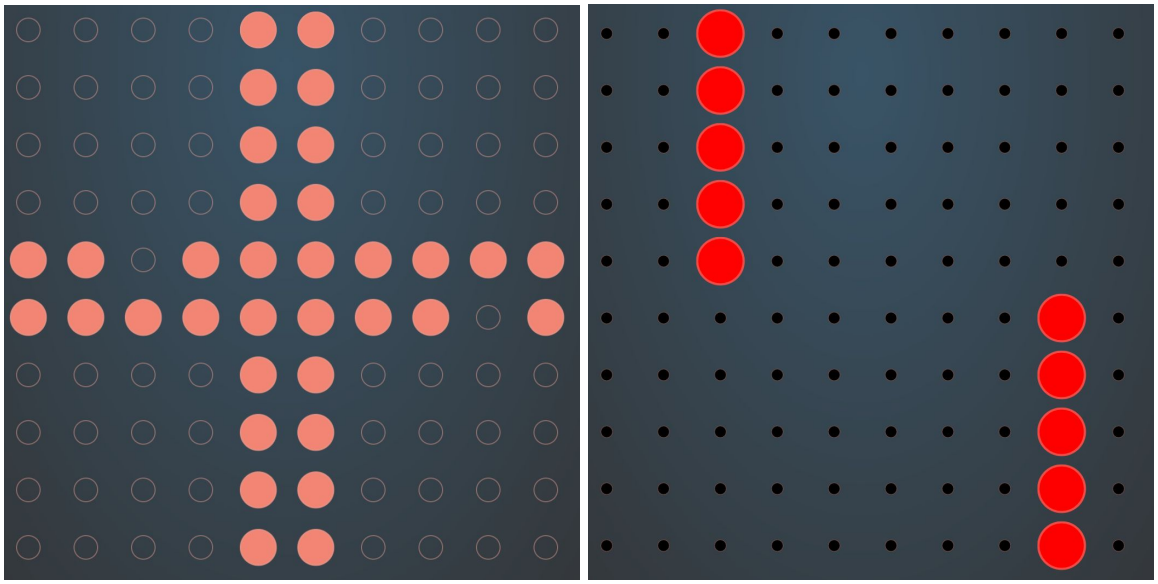
n = number of trials
k = number of successes in n trials
a = percent chance of making a type 1 error

Confidence Interval Source:
http://www.sigmazone.com/binomial_confidence_interval.htm

VISUAL ELEMENTS

Once a user hits “Calculate” the grid circles (given the list of ships left to find and the locations already shot into) are re-shaded and resized to reflect the probability of hitting a ship if a user were to call their next hit in that location. The larger the circle/ redder the circle, the higher probability. The smaller/grayer the circle, the smaller the probability. A user is then able to hover over the grid spaces to display the true probability along with the 95% confidence interval of the true probability.

An interesting visual result is that our visualization makes it easy for a user to recognize when grid spaces cannot hold a ship. For example, the images below show a user trying to find an aircraft carrier (which has length 5). Because the carrier only fits in a space of 5 units straight, given the initial starting conditions, the carrier can only be located in one of two locations (with equal probability) indicated by the heat map.



Left: There are many open spaced on the inputted board but only 2 possible locations for the carrier to be
Right: With high probability, the carrier is located either on the 5 left cells or the 5 right cells. Our visualization discards many of the “un-shot” cells as viable locations for your next shot.

The results above hold true for more complex combinations of ships and inputted boards.