Battleship and Hangman Theory

Grant Yang Atharv Goel

Fig. 1. The setup of our channel to model Battleship. Our code acts as the encoder and decoder, encoding a game state into a series of Battleship hits and misses.

I. Battleship

We can model a game of battleship as the process of encoding and transmitting a game state through a channel through which only information about Battleship moves are allowed. We define G as the complete, true configuration of the ships, as chosen by Dr. Aiyer acting as the source. Our program then acts as both the encoder and decoder, and we denote the game state received on the other side of the channel as R. Dr. Aiyer also acts as the channel, which is lossless in the case of Battleship as no lying occurs. Thus, $I(X;Y) = H(X)$ as $H(X|Y) = 0$.

In order for the channel to model the act of playing Battleship, X (and also Y) must only consist of valid answers to questions during a game of Battleship. The questions themselves (which take the form of guessing a square) are determined by the channel coding scheme (our algorithm) and thus not modeled. To design an algorithm to effectively play battleship, we merely need to design an efficient channel code for this setup.

There are a total of 30,093,975,536 possible configurations of ships in Battleship, so we have 34.8 bits to transmit. Here, we count permutations of the two 3-long ships as two separate positions, but the opposite approach is also reasonable. The 7 possible values for X are as follows: 'hit', 'miss', and 'sunk ship 1' through 'sunk ship 5'. In the best-case scenario of a uniform distribution for X, this gives $H(X) = 2.8$ bits, allowing us to transmit any game state of Battleship within 13 symbols. However, a uniform distribution for X is obviously not realistic: achieving a little less than 1 bit of information per turn is a more realistic estimate.

Because the objective of Battleship is to sink all of the opponent's ships, optimal play should involve picking a square such that the probability of getting a hit is maximized. This probability can be calculated with two algorithms, one for the early-game and another for the mid- to late-game. These algorithms make the assumption that all possible ship configurations are equally likely, and so they do not take into account ship placement strategy.

The first algorithm is random sampling. We used rejection sampling to explore the possibility space uniformly, randomly placing all five ships on random squares in a random orientation. If the configuration was consistent with the observed information up to this point (hits, misses, and ships sunk), the occupancy pattern was counted. However, more and more samples will be rejected as we reveal more information about the game state, rapidly decreasing our algorithm's performance. Although there are methods to prune away invalid configurations earlier, they all heavily bias the sample such that it is no longer uniform. Thus, we switch to another algorithm after revealing two hits on the board. Up to that point, we are able to sample over 500,000 Battleship positions in around 10 seconds in the worst cases.

Our second strategy is simply full enumeration of all possible battleship states given the current observed pattern of hits and misses. Using various optimizations, we were able to achieve an enumeration speed of roughly 100 million positions per second. The most notable optimizations were the aggressive early pruning of the search tree and the use of 128-bit bitboards to represent the occupancy of Battleship boards.

These strategies allow close to optimal play in the early-game and optimal play after switching to enumeration. The first enumerations initially take around 10 seconds, but after sinking a ship, enumerating is near instantaneous.

II. Hangman

The source is the true, current game state as it exists in Dr. Aiyer's mind. The encoder is the program because it decides how and what information is going to be communicated from the source to the receiver. The channel is Dr. Aiyer. Because there is lying, this channel is imperfect. The decoder is the program, which takes the output of the channel and decodes it to obtain the game state. The receiver is also the program, which uses a decoded game state to establish a new algorithm for the encoder in the following round.

We consider Hangman as two different subproblems. The first is picking the optimal letter given a current game. The second is updating the game after receiving information from the user. We begin with the latter.

Since the user can lie, we are uncertain about the game state. Hence, let the current game be the random variable S with probability mass function $p(s)$. Each different possible value of S now represents the simplified game state given that the user lied on a specific turn, or not at all. At the beginning of the game, there is only one possible value for S: the *truth node*. Note that the user can only lie once. Thus, as the game progresses, S gains more possible values as the *truth node*

splits into a new *truth node* and a *lie node* that assumes the user lied on that turn. However, all the *lie nodes* from previous turns can get filtered by the new information as if it was telling the truth. $p(s)$ is defined as a geometric distribution with $p = 0.25$. It is a little arbitrary, but it works.

For picking the optimal letter, we merely choose whichever gives the most expected information, across all possibilities for S. Each word has a relative frequency sourced from the web. Since the specific value of S does not matter, we condense it into a single state, with each word frequency scaled by $p(s)$ where s is the node that the word is in. Identical words in different nodes are combined such that their $p(s)$ scaled frequencies are added. The phrases are flattened as well. Thus, we obtain a single structure with a random variable W for each word, with a probability mass function obtained from the word frequencies. For a given letter, the expected information is found by summing $-p\log(p)$ over every different pattern in which the letter can appear, where p is the sum of the probabilities of the words to which the pattern corresponds. This is looped over every letter, and whichever one yields the highest expected information is guessed. Since this guessed letter yields the highest expected information, it will cut down the possible words or nodes more than any other guess. That means choosing this computed letter will also minimize the number of queries to determine the source or true phrase.