CSE 202: Project Formulation

Martakis, Alex
`amartakis@ucsd.edu A59012834`

Sharan, Mayank
`msharan@ucsd.edu A59012192`

Choudhary, Twinkle
`twchoudhary@ucsd.edu A59016512`

Gupta, Satvik
`sag005@ucsd.edu A59012252`

February 16, 2023

1 Introduction

Our project is to model the game of battleship and solve algorithmic problems associated with it. In this document, we will mathematically model the game. In addition, we present mathematical formalization of the algorithmic problems that we are attempting to solve.

Battleship is played between 2 players. The objective of the game is to sink all ships of your opponent before they are able to sink yours. The gameplay has the following phases:

1. **Setup:** Both players arrange the ships they have on their grid.
2. **Guessing:** Both players take turns guessing a grid position of the opponent each turn
   - In this phase, if a guess is where an opponent ship is present, it is a ”hit”.
   - otherwise, it is a ”miss”.
   - A ship is sunk if all grids it is located in, are hit.
   - The player who sinks all ships of the opponent first wins.

Some key components of the game to be modeled are:

1. The board
2. The ships: count and length
3. The arrangement of ships
4. Player guesses and their outcome (”hit” or ”miss”)

In Section 2, we define the basic variables required to model the game which will be used in multiple algorithmic problems. We also discuss the restrictions we impose to limit the complexity of the problems we tackle. Section 3 formalizes the algorithmic problems we will be attempting to solve. They also define any additional variables or constraints pertaining to the specific problem.

2 Game Modeling

2.1 Board Size

A typical game of battleship is played on a square board of size 10x10. However, to define more general problems and algorithms we will consider rectangular boards of arbitrary size $m \times n$, where m is the number of rows and n is the number of columns.

Without loss of generality we will assume $n \geq m$. This holds as any board with more row than columns can be rotated by 90° to get a board with more columns than rows. Thus, all configurations and actions have a rotational mapping.

Each grid position on the board will be indexed using a tuple of the form (x, y), where x is the row number and y is the column number. Further, $1 \leq x \leq m$ and $1 \leq y \leq n$.

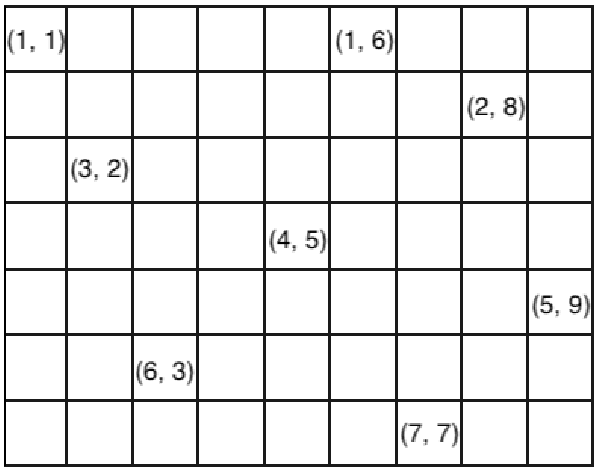

Figure 1: A rectangular board of size 7 x 9 with grid positions illustrated

2.2 Ships

A standard battleship game has five ships of length 2, 3, 3, 4, and 5. In the spirit of generalization, we will take the number of ships and their sizes as parameters. We will denote the number of ships as n_s. We will restrict each ship to be 1 grid wide, so the only dimension they have is length.

$1 \leq n_s \leq m$, as we want there to be at least one ship. The upper limit to ensure that if needed, each ship can be put on separate rows or columns. The number of ships also contributes to the nature of the game. A sparse setup will likely have a very different approach than a dense one.

The length of the ships would be provided as an array *shipLen* of length n_s, indexed from 1 to n_s. $1 \leq shipLen[i] \leq m$ and $shipLen[i] \in \mathbb{Z}^+ \ \forall 1 \leq i \leq n_s$. This is to ensure that every ship can fit within any row or column it is placed on.

2.3 Arrangement of Ships

For any given ship with identifier i its location on the grid for player p is defined as a pair of tuples $(x^p_{is}, y^p_{is}), (x^p_{ie}, y^p_{ie})$, both valid coordinates on the board as defined in Section 2.1. For simplicity, we order the tuples so that $x^p_{is} \leq x^p_{ie}$ and $y^p_{is} \leq y^p_{ie}$. Here, $x^p_{is} = x^p_{ie}$ for a placement along a row or $y^p_{is} = y^p_{ie}$ for a placement along a column. We do not allow diagonal placement of ships as per the rules of the game. In case of a row placement $y^p_{ie} - y^p_{is} + 1 = shipLen[i]$ and in case of a column placement $x^p_{ie} - x^p_{is} + 1 = shipLen[i]$. These tuples for each ship are stored in lists of tuples $shipS_p$ containing all (x^p_{is}, y^p_{is}) tuples and $shipE_p$ containing all (x^p_{ie}, y^p_{ie}) tuples. Both arrays are indexed by the ship identifier i.

Ships are not allowed to overlap so there does **NOT** exists any tuple (x, y) such that for any 2 ships i and j the following conditions are all true:

$$x^p_{is} \leq x \leq x^p_{ie}$$
$$x^p_{js} \leq x \leq x^p_{je}$$
$$y^p_{is} \leq y \leq y^p_{ie}$$
$$y^p_{js} \leq y \leq y^p_{je}$$

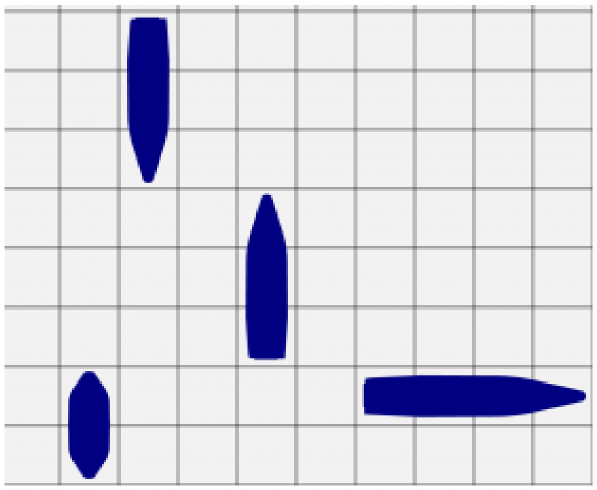

Figure 2: A sample arrangement of ships

2.4 Guess outcomes

We maintain 2 two-dimensional matrices to represent the guess status. For players p and q, these matrices are $Guess_p$ and $Guess_q$. Both matrices are of size $m \times n$ in correspondence to the board and are indexed starting from 1 so that the position tuples can be used as indices. $Guess_p[(x, y)]$ will provide the status on player q's grid of the guess by p at location (x, y) and vice versa.

Both matrices are initialized with all "U"s to indicate unexplored. As the game progresses in phase 2 the outcomes of all guesses by player p are updated in $Guess_p$ and by player q are updated in $Guess_q$. The value is updated to "H" in the event of a hit and to "M" in the event of a miss.

Both players have access to both matrices at all times as the information in them is available to both of them as the game progresses. We will not be storing the order of the guesses as it introduces storage complexity without adding any significant information and is not a reasonably expected for human players to be able to track.

3 Algorithmic Problems

3.1 Finding the first hit

3.1.1 Problem Statement

We want to design a strategy that given a state of the game provides a sequence of guesses resulting in a hit. This can be used at the start of the game to locate the first ship or once a ship is sunk to find the next one. We define the problem with respect to the strategy of one player. This can be used by both players given the state of their corresponding $Guess$ matrix.

We find a sequence of guesses that results in a hit from a given game state.

3.1.2 Mathematical formulation

Instance

- Grid size: $m \times n$ (section 2.1)
- Ship information: $shipLen$ of length n_s (section 2.2)
- Guess State Matrix for guessing player: $Guess_p$ (section 2.4)

Since the opponent's ship arrangement (Section 2.3) is unknown, it is not an input to our algorithm.

Solution

A sequence S of guesses made by player p: $[(x_1, y_1), (x_2, y_2)..(x_i, y_i)..(x_k, y_k)]$

Constraints

Guess $S[i] = (x_i, y_i)$ is a valid grid position, i.e. $1 \leq x_i \leq m, 1 \leq y_i \leq n$

All guesses are unique $\forall i, j \; i \neq j \implies S[i] \neq S[j]$

None of the guesses have already been made in the initial guess matrix, $\forall i, Guess_p[S[i]] = U$

The guesses in S are made in order by p

$Guess_p[S[i]] = M \; \forall 1 \leq i \leq k-1$

$Guess_p[(x_k, y_k)] = H$. This is because otherwise we can find a solution of smaller size that finds the first hit.

Objective

minimize k

3.2 Sink the Ship

3.2.1 Problem Statement

Given a state in which a player p has a successful hit we would like to sink the ship that is hit in the minimum number of guesses.

This problem has 2 variants one in which player q will provide the length of the ship that has been hit and the other in which this information is not shared with player p. We will refer to the variant with the length information as Variant 1 and without the information as Variant 2.

3.2.2 Mathematical formulation

Instance

- Grid size: $m \times n$ (section 2.1)
- Guess State Matrix for guessing player: $Guess_p$ (section 2.4)
- Ship information: $shipLen$ of length n_s (section 2.2)
- Identifier of ship that is hit: i [Input for variant 1 only]

Since the opponent's ship arrangement (Section 2.3) is unknown, it is not an input to our algorithm.

Solution

A sequence S of guesses made by player p: $[(x_1, y_1), (x_2, y_2)..(x_i, y_i)..(x_k, y_k)]$

Constraints

Guess $S[i] = (x_i, y_i)$ is a valid grid position, i.e. $1 \leq x_i \leq m, 1 \leq y_i \leq n$

All guesses are unique $\forall i, j \; i \neq j \implies S[i] \neq S[j]$

None of the guesses have already been made in the initial guess matrix, $\forall i, Guess_p[S[i]] = U$

The guesses in S are made in order by p

$$\forall (x, y),\ shipS_q[i] \leq (x, y) \leq shipE_q[i] \implies \exists j S[j] = (x, y)$$

In other words guesses should sink the ship. In Variant 2, i refers to the ship that was hit but is unknown to player p.

Also, $shipS_q[i] \leq S[k] \leq shipE_q[i]$, the final guess should be a hit on the ship which we are trying to sink. Combined with the previous constraint this is also the hit that sinks the ship. This is because otherwise we can find a solution of smaller size that sinks the ship.

Objective

minimize k

Note

As per the feedback to our project proposal we have decided to not attempt directly solving the problem of identifying the initial placement of ships. However, we may include any interesting results pertaining to this if we derive them during our efforts. The definition of a valid ship arrangement in section 2.3 can be considered an informal mathematical rendition of this problem that we will use to include any analysis.