Deadpool - Battleship

Xavier Chassin

June 12, 2016

Notations

s Side length of the board.

1 Introduction

This document presents the *Deadpool Battleship* project. The focus is on the algorithms used to play a *Battleship* game, as well as their implementation. Those will hopefully be implemented in different languages:

- Golang
- Python
- And potentially others (C++, Haskell...)

2 Playing randomly

To setup game and have a working implementation of the server, I needed a first player. The easiest, non completely trivial algorithm is the random one. It randomly picks a tile amongst all the tiles on which it has not fired yet.

3 Scoring the tiles

The random player is helpful to get the server started, however it is not exactly funny from a algorithm point of view. An idea of a battleship algorithm would be to score each tile and then fire on the tile with the highest score.

3.1 Distance based

To score a tile, we can use the distance from that tile the closest known (i.e. fired upon) tile. When playing a battleship game, if you fire on a tile that happens to only be the sea, then the adjacent tiles are less likely to be ship tiles.

3.1.1 Ship adjacent tiles

The obvious drawback of such a scoring function is that if we actually fire on a ship, the adjacent tiles will have a minimal score and hence will not be fired upon. To overcome this drawback, we set the score $s = \max_{t \in \mathcal{T}} s_t$ to the tiles adjacent to a ship.

3.1.2 A bit of randomness

In order to avoid having tiles with the same score, we add a random "error" to the distance:

$$s_t = \min_{t \in (F)} d(t, t') + \epsilon$$

where $\epsilon \sim \mathcal{N}(0, 1)$. This also has the advantage of enhancing exploration of the board.