Algorithmic Naval Warfare: Implementing and Solving Battleship Grids

Group - 026

January 13, 2024

Contents

Abstract

This project report presents an algorithm designed to generate and solve Battleship game grids. By programmatically placing ships on a grid and systematically solving it, this project demonstrates an intersection of game theory, algorithm development, and problem-solving in a simulated environment. The results showcase the efficiency of the algorithm in terms of hit ratio and the number of cells checked.

1 Introduction

1.1 Background

The game of Battleship is a classic naval strategy game where players place ships on a grid and take turns guessing the locations of the opponent's ships. This project digitizes a part of this experience by implementing an algorithm that not only generates these grids but also solves them.

1.2 Objective

The primary aim is to create a Python-based solution that can autonomously generate a Battleship grid with ships placed in random orientations and positions, and then solve this grid by identifying the locations of all ships.

1.3 Significance

Understanding the algorithmic approach to such a problem can have broader implications in understanding computational problem-solving, offering insights into efficient algorithm design and strategic thinking.

2 Methodology

2.1 Grid Generation

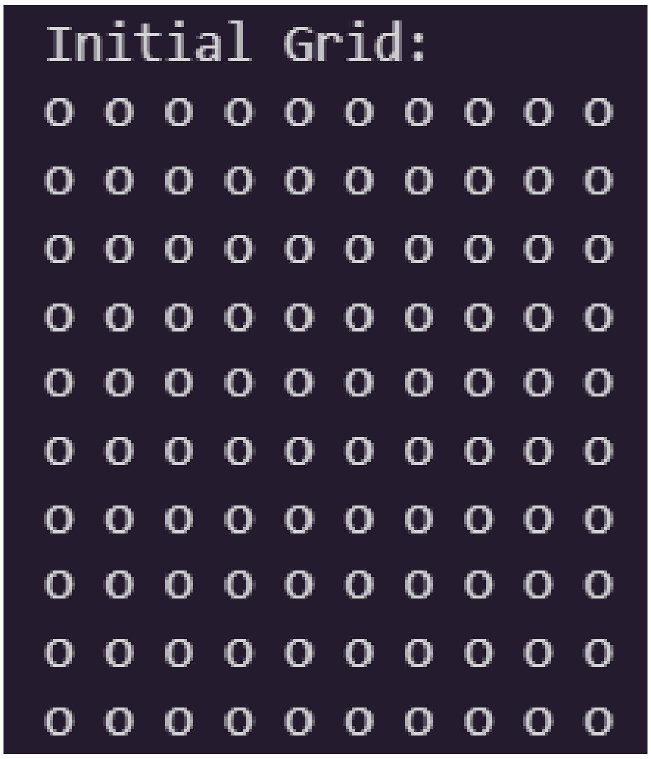
The 'create_battleship_grid(size=10)' function initializes a 10x10 grid and randomly places five types of ships on it. Each ship is defined by its size and type, such as the 5-cell Aircraft Carrier or the 2-cell Destroyer. The function ensures no overlap or out-of-bound placements.

2.2 Grid Solving

The 'solve_battleship(grid)' function systematically checks each cell of the grid. It marks hits when a ship part is found and misses otherwise. The process continues until all ship parts are hit, keeping track of the number of attempts made.

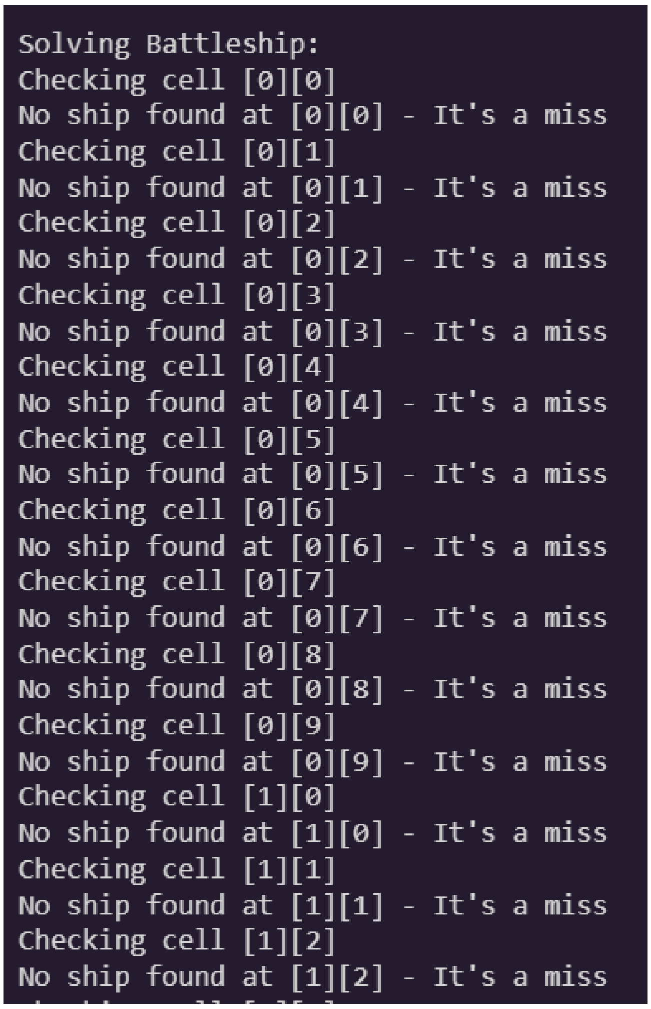
3 Results

A sample generated grid was successfully populated with all ships randomly placed. The solver algorithm managed to identify all ship locations. It achieved a hit ratio of approximately 17% (17 hits out of 100 cells), with 83 misses. The systematic approach ensured that no cell was checked more than once.



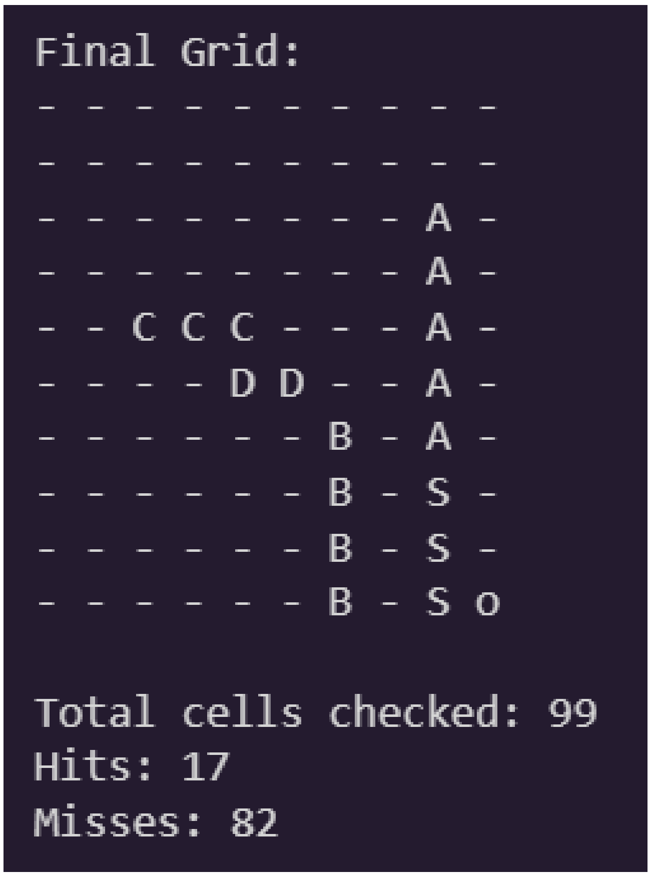
```
Initial Grid:
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
O O O O O O O O O O
```

Figure 1: Initial Grid



```
Solving Battleship:
Checking cell [0][0]
No ship found at [0][0] - It's a miss
Checking cell [0][1]
No ship found at [0][1] - It's a miss
Checking cell [0][2]
No ship found at [0][2] - It's a miss
Checking cell [0][3]
No ship found at [0][3] - It's a miss
Checking cell [0][4]
No ship found at [0][4] - It's a miss
Checking cell [0][5]
No ship found at [0][5] - It's a miss
Checking cell [0][6]
No ship found at [0][6] - It's a miss
Checking cell [0][7]
No ship found at [0][7] - It's a miss
Checking cell [0][8]
No ship found at [0][8] - It's a miss
Checking cell [0][9]
No ship found at [0][9] - It's a miss
Checking cell [1][0]
No ship found at [1][0] - It's a miss
Checking cell [1][1]
No ship found at [1][1] - It's a miss
Checking cell [1][2]
No ship found at [1][2] - It's a miss
```

Figure 2: Solving



```
Final Grid:
- - - - - - - - - -
- - - - - - - - - -
- - - - - - - - A -
- - - - - - - - A -
- - C C C - - - A -
- - - - D D - - A -
- - - - - - B - A -
- - - - - - B - S -
- - - - - - B - S -
- - - - - - B - S o

Total cells checked: 99
Hits: 17
Misses: 82
```

Figure 3: Final Grid

4 Discussion

4.1 Algorithm Efficiency

The algorithm efficiently solves any given Battleship grid. However, it follows a brute-force approach, checking each cell without any strategic inference from previous hits.

4.2 Potential Improvements

Future improvements could include implementing a more intelligent hit-follow-up strategy, potentially reducing the number of checks required to solve the grid.

5 Conclusion

This project successfully demonstrates an algorithmic approach to generating and solving Battleship grids. It reflects the potential of simple algorithms in solving complex problems and lays the groundwork for more sophisticated problem-solving strategies in similar contexts.

6 References

1. ”Battleship (game)” - Wikipedia, The Free Encyclopedia.

2. ”Algorithms” - by Robert Sedgewick and Kevin Wayne.

7 Appendix: Source Code

7.1 Battleship Grid Generator and Solver

The following Python script includes two main functions: ‘create_battleship_grid‘, which generates a 10x10 grid with battleships placed randomly, and ‘solve_battleship‘, which systematically checks each cell to find and hit all ships.

```python

import random

def create_battleship_grid(size=10):
    # Function to create a battleship grid with random ship placements
    # ...

def display_grid(grid):
    # Function to display the grid
    # ...

def solve_battleship(grid):
    # Function to solve the battleship grid by finding all ships
    # ...

# Generate and solve the battleship grid
battleship_grid = create_battleship_grid()
print("Initial Grid:")
display_grid(battleship_grid)
print("Solving Battleship:")
solve_battleship(battleship_grid)
```

Note: The code includes random ship placement and a systematic approach to solve the generated grid. The grid solver iterates through each cell, marking hits and misses, until all ships are found.