RMIT UNIVERSITY

COSC 2123/1285 Algorithms and Analysis
Assignment 2: Battleship Game

	Assessment Type	Individual assignment. Submit online via Canvas → Assignments → Assignment 2. Clarifications/Updates/FAQ: check Ed Discussion Forum → Assignment 2: General Discussion.
	Due Date	Week 12 Friday, May 26, 11:59pm
	Marks	30

1 Objectives

In this assignment, we focus on designing and analysing different strategies for the *Battleship game*. Please note that this assignment **does not** require any implementation, but you are very welcome to implement solutions to test them out.

2 Background

Battleship is a classic two player game. The game consists of two players, and each player has a number of ships and a 2D grid. The players place their ships onto the grid, and these ships takes up a number of cells. The ships cannot overlap with each other. Each player takes turn at guessing a cell to fire at in their opponent's grid. If that cell contains part of a ship, the player gets a hit. If every part of a ship has been hit, then it is sunk and the owner of the ship will announce the name of the ship sunk (see ships section below for ships in the standard game). The aim of the game is to sink all of your opponent's ships before they sink all of yours. For more details, see https://en.wikipedia.org/wiki/Battleship_(game).

Traditionally, Battleship is played between human players. In this assignment, you will design and develop[1] algorithms to play Battleship, that uses a variety of the algorithmic paradigms we have covered in class. It will also give you a taste of artificial intelligence (AI), as algorithms is an important component of AI.

To help you understand the tasks better, we will use the example illustrated in Figure 1, which is a Battleship game played on a 10 by 10 grid.

[1] Stressing there is no need to implement.

	0	1	2	3	4	5	6	7	8	9
9										
8										
7										
6										
5										
4										
3										
2										
1										
0										

(a) A 10 by 10 grid.

	0	1	2	3	4	5	6	7	8	9
9										
8										
7										
6										
5										
4										
3										
2										
1										
0										

(b) A 10 by 10 grid with a submarine and a frigate placed.

Figure 1: Running example used to illustrate the concepts in the tasks.

2.1 Ships

In the standard Battleship game, there are five ships available for each side. For this assignment, we have six ships that are different from the standard Battleship game and have dimensions as follows:

Name	Dimensions
KoKo	1 by 2 cells
Cruiser	2 by 2 cells
Submarine	1 by 3 cells
Frigate	1 by 4 cells
Aircraft Carrier	2 by 3 cells
Unicorn	3 by 3 cells (+ shape, occupying 5 cells)

2.2 Player Strategies for Guessing Cells

There are several different strategies for guessing a cell in the Battleship game, apart from random guessing which is the simplest and most basic strategy, where the player randomly chooses a cell on the game board to make a guess. We can divide the guessing process into two parts: *hunting* mode, where the player is seeking opponent's ships, and *targeting* mode, where once there is a hit, the player tries to sink the partially hit/damaged ship. The following two strategies adopt the hunting & targeting modes:

2.2.1 Strategy I

When hunting for a hit, the player can utilise the fact that the ships are at least 2 cells long and use the partiy principle. See Figure 2a. As ships listed in Section 2.1 are at least of length 2, the player do not need to fire at every cell to ensure finding the opponent's ships. Rather, there is a need to just fire at every second square (Figure 2a). Hence, when hunting for one of the opponent's ships, it can now randomly select a cell from a checkboard type of pattern.
Once a cell registers a hit, the player knows the rest of the ship must be in one of the four adjacent cells, as highlighted as orange circles in Figure 2b. The player seeks to destroy the ship before moving on, hence will try to fire at those four possible cells first (assuming they haven't been fired upon, if they have, then no need to fire at a cell twice). Once all possible targeting cells for any hit have been exhausted, the player can be sure to have sunk the ship(s) (can be more than one if ships are adjacent to each other) and it returns to the hunting mode until it finds the next ship.

	0	1	2	3	4	5	6	7	8	9
9	X		X		X		X		X	
8		X		X		X		X		X
7	X		X		X		X		X	
6		X		X		X		X		X
5	X		X		X		X		X	
4		X		X		X		X		X
3	X		X		X		X		X	
2		X		X		X		X		X
1	X		X		X		X		X	
0		X		X		X		X		X

(a) Parity idea. The player doesn't need to fire at every cell to guarantee all ships are at least hit. The cells with Xs (can be the other alternative, if the top left most X is at (9,1)), are the ones to try.

	0	1	2	3	4	5	6	7	8	9
9										
8		X			X					
7										
6				X						
5						O				
4		X			O	X	O			
3						O				
2		X								
1									X	
0						X				

(b) In targeting mode. Red X denote a hit, and the next four cells to target are the ones with orange Os.

Figure 2: Illustration of the parity principle and the targeting mode.

	0	1	2	3	4	5	6	7	8	9
9	+									
8										
7										
6										
5										
4										
3										
2										
1										
0										

(a) The number of configurations (2) that the frigate placed into top left cell (highlighted with the '+').

	0	1	2	3	4	5	6	7	8	9
9										
8										
7										
6										
5					+					
4										
3										
2										
1										
0										

(b) The number of configurations (10) that the frigate placed into the cell at the centre (highlighted with the '+').

Figure 3: Illustration of the idea behind counting the number of ship placements that can go through the '+' cell.

2.2.2 Strategy II

In Strategy I the player assumes every cell is as likely to contain a ship. But this is unlikely to be true. For example, consider the frigate and a 10 by 10 grid. It can only be in two placement configurations if placed in top left corner (see Figure 3a), but in 8 different placement configurations if part of it occupies one of the cell in the centre, e.g., cell (4,4) (see Figure 3b). Hence, assuming the opponent randomly places ships (typically they do not, but that is beyond this course, as we are going towards game theory and more advanced AI), it is more likely to find the frigate occupying one of the cells in the centre. This exercise can be repeated for all ships, and for each ship, we end up with a count of the number of ship configurations that can occupy that cell. The cell with the highest total count over all ships is the one most likely to contain a ship.

In the hunt mode, this type of player will select from those cells yet to be fired upon, the one with the highest possible ship configuration count (if there are several, randomly select one). If there is a miss, then the count of that cell and neighbouring cells (because the shot is missed, it means there is not any ship that can occupy that cell, so the player needs to update its count and the neighbouring ones, as the count of neighbour ones may depend on a ship being upon to fit onto the fired upon cell). If the shot is a hit, the player goes to targeting mode. In targeting mode, the player makes use of the fact

that there has been a hit to calculate which adjacent cell is the most likely to contain a ship (with the highest configuration count). Using the same counting method as the hunting mode, the player can calculate the number of possible ship configurations that pass through the hit cell (remember previous misses, previously sunk ships and grid boundaries should be considered as obstacles and taken into account). With every subsequent miss or hit, the counts are updated correspondingly and the player repeats firing at an adjacent cell with the highest count. Once a ship is sunk, the counts of the whole grid must be updated to reflect this ship is no longer in play. When the player has sunk the ship(s), then it goes back to hunting mode.

2.3 Writing pseudocode

In terms of writing pseudocode, the following apply:

- To fire at a specific location, use `fire(board[y][x])` to shoot at specific location (you may use other unambiguous variations such as `shoot()` if you prefer).
- After shooting at a specific cell on the board, the sets M, H and, if needed,S_r will automatically be updated.
- The time complexity refers to the entire algorithm, not just the amount of shots made.
- For the sets M, H and S_r, valid operations are that expected of regular sets such as `contains(item)`, `isEmpty()`, etc. You may iterate over all items in the set (but these will be unordered).
- You may call other functions given a reasonable algorithm already exists and their functionality is not ambiguous such as `sort(array[])` or `random(0, 10)`. External function calls should clearly have their time complexity listed.
  - Unacceptable uses include substitution of pseudocode expected to be written such as `fireAllCellsForCol()` (however this is allowed if you define your own function).

3 Tasks

The assignment is broken up into a number of tasks, to help you progressively complete the assignment. For all the tasks you can assume that the board is an N by N grid, S is the set of ships listed in Section 2.1, and f is a function that maps every ship to its width and height, e.g., $f(Frigate) = (1, 4)$ and the player has a single shot at every turn. For all the tasks, you are required to use the generic notation when writing the pseudo code or explaining the algorithm. If asked for the complexity analysis, worst-case complexity is assumed.

1. Write pseudo code for the random guessing strategy in the function below. Recall that this must work for any generic $N \times N$ board, set S and not just the 10×10 example shown above. What is the complexity of this algorithm with regard to the input parameters? Briefly justify your answer. [1 mark]

Algorithm RandomStrategy
Input board$[0, 1, ..., n-1][0, 1, ..., n-1]$, set of remaining ships, $S_r \subseteq S$, set of *miss* coordinates, M, set of *hit* coordinates, H.

Please add your solution here.

---

2. For Strategy I, what is the design paradigm, e.g., brute force, decrease & conquer, divide & conquer, etc., for each of the hunting and targeting modes? Justify your answer. Write the pseudo code for each mode, using the two functions below as the function signatures. Recall that this must work for any generic $N \times N$ board, set S and not just the 10×10 example shown above. What is the complexity of each algorithm with regard to the input parameters? Justify your answer.[3 marks]

Algorithm StrategyI-Hunt
Input board$[0, 1, ..., n-1][0, 1, ..., n-1]$, set of remaining ships, $S_r \subseteq S$, set of *miss* coordinates, M, set of *hit* coordinates, H.

Please add your solution here.

Algorithm StrategyI-Target
Input board$[0, 1, ..., n-1][0, 1, ..., n-1]$, set of remaining ships, $S_r \subseteq S$, set of *miss* coordinates, M, set of *hit* coordinates, H, the most recently *hit* coordinate, h.

Please add your solution here.

---

3. What is the underlying design paradigm, e.g., brute force, decrease & conquer, divide & conquer, etc., of Strategy II? Justify your answer. Write the pseudo code for each mode. Recall that this must work for any generic $N \times N$ board, set S and not just the 10×10 example shown above. What is the complexity of each algorithm with regard to the input parameters? Justify your answer. Analyse (in plain English) if and how the complexity changes if the ships can have any arbitrary shape. [5 marks]

Algorithm StrategyII-Hunt
Input board$[0, 1, ..., n-1][0, 1, ..., n-1]$, set of remaining ships, $S_r \subseteq S$, set of *miss* coordinates, M, set of *hit* coordinates, H.

Please add your solution here.

Algorithm StrategyII-Target
Input board$[0, 1, ..., n-1][0, 1, ..., n-1]$, set of remaining ships, $S_r \subseteq S$, set of *miss* coordinates, M, set of *hit* coordinates, H, the most recently *hit* coordinate, h.

Please add your solution here.

---

4. Demonstrate your understanding of Strategy II using the example that will be provided to you. You are required to (i) compute the probabilities for the remaining cells on the board, (ii) choose the cell with the highest probability (in case of a tie, choose the top leftmost cell among the candidates) as the player's next guess, and (iii) recompute the probabilities based on the miss/hit feedback. [3 marks]

5. So far the assumptions was that when a ship is sunk, the opponent will indicate which ship of theirs have been sunk and the player can make use of this fact to improve the guessing strategy. Now assume that the opponent does not announce the specific ship that is sunk and only announces 'sunk' once a ship is fully hit. Design an algorithm that finds the number and

name of the ships that are sunk at any given stage of the game. Discuss the data structures and analyse the complexity of your proposed solution. **Please note that unlike Tasks 1 to 3, the ships cannot be touching for this task.** [5 marks]

Algorithm SunkShips
Input board$[0, 1, ..., n-1][0, 1, ..., n-1]$, set of ships, S, set of *miss* coordinates, M, set of *hit* coordinates, H.

Please add your solution here.

---

6. Task 5 was submitted to ChatGPT for its assistance. Critically analyse the following generated response in terms of (i) the correctness of the algorithm, (ii) the accuracy of the complexity analysis. If you are supportive of the proposed solution (or a modified version of it), provide the pseudo code and analyse the complexity of the solution. If not, provide cases to support your argument. [6 marks]

   To identify the number and name of the ships that are sunk in the Battleship game at any given snapshot, we can use a modified version of the flood fill algorithm.

   Here's a high-level description of the algorithm:

   1. *Initialize an empty list of sunk ships.*
   2. *For each hit cell on the board:*
      a. *If the hit cell has already been visited, skip it.*
      b. *Otherwise, perform a flood fill search starting from the hit cell to identify the size and shape of the ship.*
      c. *If the ship is completely surrounded by hit cells, add it to the list of sunk ships.*
   3. *Return the list of sunk ships.*

   The flood fill search is performed recursively, starting from the hit cell and exploring its neighbors until all cells that belong to the ship are visited. The search is terminated when a boundary is reached, i.e., when the neighboring cells are either already visited, a miss, or outside the board.

   To improve the performance of the algorithm, we can use a hash table to keep track of the visited cells and avoid redundant searches. We can also use a union-find data structure to efficiently group cells that belong to the same ship and determine if a ship is completely surrounded by hit cells.

   The time complexity of the algorithm depends on the size of the game board and the number of hit cells, but it is typically linear or sublinear in the number of cells on the board. The space complexity is also proportional to the number of hit cells and can be further reduced by using a bit vector or a sparse matrix to represent the board. Overall, this algorithm can efficiently identify the sunk ships in the Battleship game at any given snapshot.

7. *[ADVANCED]* Assume a version of the Battleship game where the player does not have any information about the number and shape of the ships. Instead the following are known:

   a. The ships occupy a total of 20 cells in the board.
   b. A ship cannot be a single cell.
   c. A ship is composed of a combination of horizontal and/or vertical cells.
   d. A ship cannot be teaching any other ship.

Design an algorithm that can find the number of ships that are sunk at any given stage of the game. Provide the pseudo code as well as the complexity analysis for your proposed solution. [7 marks]

4 Important Notes

- If you are asked to design an algorithm, you need to describe it in plain English first, say a paragraph, and then provide an **unambiguous** pseudo code, unless specified otherwise. The description must include enough details to understand how the algorithm runs and what the complexity is. All algorithm descriptions and pseudo codes required should be at most half a page and at most a page in length, for tasks 1-3 and tasks 4-7 respectively. Please remember that **worst-case** complexity is assumed unless specified otherwise. The following sample pseudo code is considered unambiguous, i.e., the inputs, output and the steps are all clearly described:

Algorithm Finding the minimum value in an array
Input Array A of size n.
Output Minimum value in the array.

```
1: procedure FindMinimum(A)
2:     minVal ← A[0]
3:     for i ← 1 to n − 1 do
4:         if A[i] < minVal then
5:             minVal ← A[i]
6:         end if
7:     end for
8:     return minVal
9: end procedure
```

- Marks are given based on **correctness, conciseness** (with page limits), and **clarity** of your answers. If the marker thinks that the answer is completely not understandable, a zero mark might be given. If correct, ambiguous solutions may still receive a deduction of 0.5 mark for the lack of clarity. Marks for task 5 and 7 also reflects the comparative **efficiency** of the proposed solution, i.e., the more efficient the solution relative to other proposed solutions, the higher the mark.

- Please **do NOT include the problem statements in your submission** because this may increase Turnitin's similarity scores significantly.

- This is an **individual assignment**. While you are encouraged to seek clarifications for questions on Ed Forum, please do NOT discuss solutions or post hints leading to solutions.

- In the submission (your PDF file), you will be required to certify that the submitted solution represents your own work only by agreeing to the following statement:

> I certify that this is all my own original work. If I took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in my submission. I will show that I agree to this honour code by typing "Yes":

5 Submission

- Submit a single PDF file with **maximum 6 pages excluding appendix and references** named Assign2-s12345.pdf (REPLACING with your student number) and submit in Assignment 2 page on Canvas.
- Make sure you submit the LATEST/CORRECT version of your code/report well before the deadline to avoid congestion, and VERIFY that the uploading has been done successfully. We will mark the version submitted last before the deadline. Any replacement after the deadline has passed will be marked with the penalty applied (3 marks per day).

5.1 Clarification to Specifications & Submissions

Please periodically check the assignment's Updates and FAQs page on the Discussion Forum for important aspects of the assignment including clarifications of concepts and requirements, typos and errors, as well as submission.

6 Assessment

The assignment will be marked out of 30. Late submissions will incur a deduction of 3 marks per day, and NO submissions will be accepted 5 days beyond the due date and afterwards will attract a mark of 0.

7 Plagiarism Policy

University Policy on Academic Honesty and Plagiarism: You are reminded that all submitted assignment work in this subject is to be the work of you and your partner. It should not be shared with other individuals. **Multiple automated similarity checking software will be used to compare submissions**. It is University policy that cheating by students in any form is not permitted, and that work submitted for assessment purposes must be the independent work of the students concerned. Plagiarism of any form may result in zero marks being given for this assessment and result in disciplinary action.
For more details, please see the policy at http://www1.rmit.edu.au/students/academic-integrity.

8 Getting Help

There are multiple venues to get help. There are weekly lectorial Q&A sessions as well as consultation sessions. We will also be posting common questions on the Assignment 2 Q&A section on Ed Discussion Forum and we encourage you to check regularly and participate in the discussions. However, please **refrain from posting solutions**.