Python Assignment 2: Battleship Game

Deadline: (Tuesday) February 14th, at 23:55
Worth 30% of the course

1 Code of Conduct

All assignments are graded, meaning we expect you to adhere to the academic integrity standards of Yale-NUS College. To avoid any confusion regarding this, we will briefly state what is and isn't allowed when working on an assignment.

Any documents and program code that you submit must be fully written by yourself. You can, of course, discuss your work with fellow students, as long as these discussions are restricted to general solution techniques. Put differently, these discussions should not be about concrete code you are writing, nor about specific results you wish to submit. When discussing an assignment with others, this should never lead to you possessing the complete or partial solution of others, regardless of whether the solution is in paper or digital form, and independent of who made the solution, meaning you are also not allowed to possess solutions by someone from a different year or course, by someone from another university, or code from the Internet, etc. This also implies that there is never a valid reason to share your code with fellow students, and that there is no valid reason to publish your code online in any form.

Every student is responsible for the work they submit. If there is any doubt during the grading about whether a student created the assignment themselves (e.g. if the solution matches that of others), we reserve the option to let the student explain why this is the case. In case doubts remain, or we decide to directly escalate the issue, the suspected violations will be reported to the academic administration according to the policies of Yale-NUS.

2 Introduction

Battleship (also Battleships or Sea Battle) is a guessing game for two players. It is known worldwide as a pencil and paper game which dates from World War I. It was published by various companies as a pad-and-pencil game in the 1930s, and was released as a plastic board game by Milton Bradley in 1967.

The goal of this assignment is to implement a simplified version of the Battleship game in Python. This will give you practice in working with lists, decision structures, iterations and loops, etc. Additionally, it is a good exercise in decomposing a larger problem into smaller and more manageable parts.

The Battleship game is played on a two-dimensional board. The board is typically square (usually 10x10 cells) and the individual cells in the grid are identified by letters and numbers. In this simplified version of Battleship, only one player plays against the computer. At the start of the game, the computer secretly places one (hidden) ship with the size of **3** cells/blocks on the board in a random location and with a random orientation, either horizontally or vertically. As the location of the ship is concealed from the player, the goal of the player is to "destroy" the ship with the least number of guesses. Hereby, the player takes turns by entering a coordinate of the target cell. If it is a hit, the cell is marked by an 'X', otherwise by a '#' for a miss. When all cells of the ship are guessed by the player, the ship is sunk and the game is won.

Figure 1 shows an example of the Battleship game on a 10x10 grid.

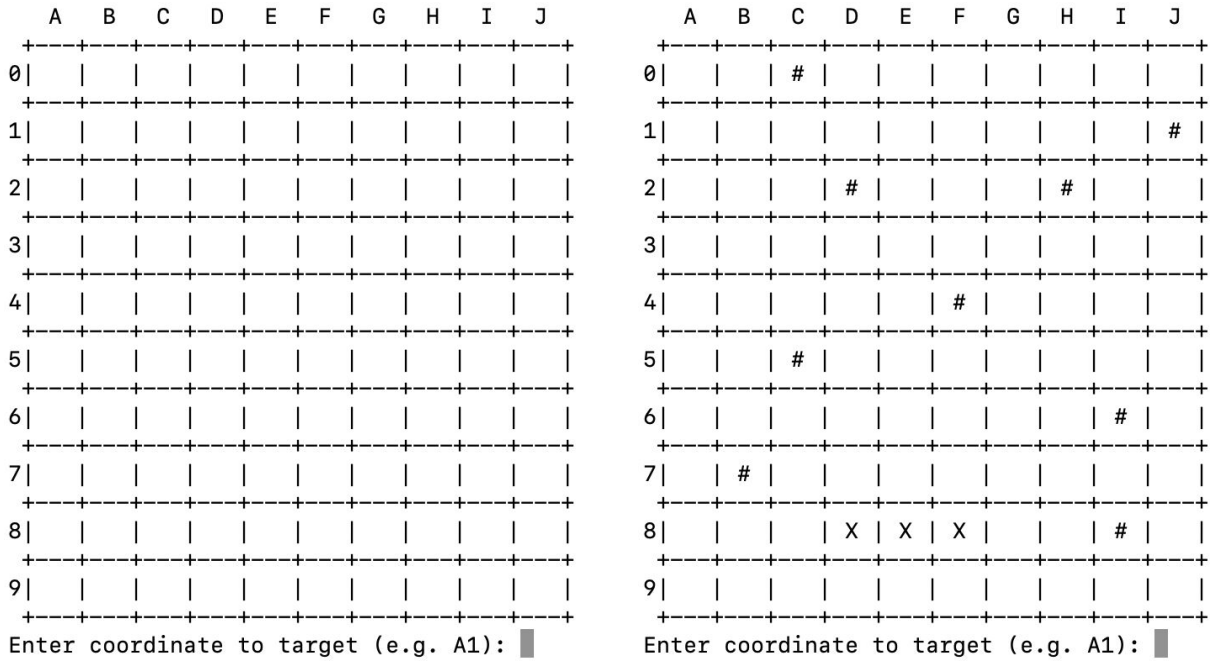

Figure 1: Example output after the start of the game and after multiple turns with 3 hits at D8, E8, F8

3 Implementation

The implementation of this game mainly consists of two phases:

3.1 Initialization phase:

At the start of the game, an (empty) board is created, where columns are identified by a letter and rows by a number (see Figure 1). The computer then randomly places a ship on the board, where the location of the ship is described by a random row, column, and orientation. As the ship occupies

a number of consecutive cells on the board, the random row and column is used as the first cell of the ship and then extends either vertically (down) or horizontally (right) based on the length of the ship. In order to randomly select a row, column, and orientation of the ship, use the randint() function of the random module (as discussed in class). When drawing random numbers for the first cell of the ship, make sure that a) the row and column is within the board dimensions and b) the ship can actually be placed completely within the board dimensions, i.e. no cell of the ship exceeds the board dimensions

As the location of the ship is concealed from the user, you have to find a solution to store the cells that are occupied by the ship without revealing them to the player, for example use another board that holds the locations of the ship which is not shown to the user and only used to verify a hit or miss

After creating the board, the board should be printed on the screen so that the user gets an idea on how the board looks like

Please note the board dimensions and the size of the ship are declared as constant (global) variables and that the program dynamically creates a board based on these values. In other words, do not "hardcode" the board dimensions, size of the ship, etc. This allows you to easily change these values without modifying multiple lines of your code. The code snippet below shows an example of the first lines of your code:

```
1 import random              # import the random module
2
3 SHIP_SIZE = 7              # constant variable for the size of the ship
4 DIMENSION = 10             # constant variable for the size of the board (square)
5
6 # Create Board DIMENSION X DIMENSION
7
8 # randomly place the ship. Use random.randint() to draw numbers
9
10 # display the board
```

3.2 Game phase:

After creating the board, the player is asked to make a guess by entering a coordinate (location) within the board dimensions. The coordinate consists of the column (a letter) followed by the row (a digit). For example, valid coordinates are B4, C1, A6, etc. Notice that the column letter is case-sensitive. Also make sure that the input clearly describes the format of the expected input. In case the player enters an invalid input or an invalid coordinate, the program informs the player about this and asks for a new coordinate. Examples for invalid input are AB, A 6, 1A, 32, A;5, etc. or coordinates out of the board dimensions

After a valid user input, the program places the guess on the board, where a hit is denoted as an 'X' and a miss as a '#'. The program must also maintain a score indicating the number of guesses

After each turn, the program must check if the game is won, i.e. the entire ship is sunk. If so, the program must inform the user that the game is over, display the score and terminate. Otherwise, the board should be printed reflecting the updated cells and ask for another input. Before printing the updated board, the screen should be cleared by using the following command (add line 2 and 6 to your code at the respective locations):

```
1 import random
2 import os
3
4 # Your amazing code here…..
5
6 os.system('cls') or os.system('clear')
```

In order to keep the game logic simple, we restrict our implementation to the following:

1. The minimum and maximum board dimensions are 4x4 and 10x10 cells, respectively. Your code does not have to deal with smaller or larger dimensions or dimensions that are not square.

2. The ship length is between 3 and 7 cells.

4 Submission

Please note: Your submission must have set the board dimensions to 10x10 and the ship length to 7 cells.

The deadline of this assignment is 12 days, **the firm deadline is on (Tuesday) February 14th, at 23:55**. Please submit/upload to Canvas one .py file containing your **Python** program. Submissions via email are not accepted.

Note that your solution must work using Python.

In case your code does not work using Python, your submission will not be graded.

Late submissions will be penalized by 50% per 24 hours.

5 Grading (Name 😊)_____

Description	Score (/30)	Your Score
Initialization of the board and usage of global variables (constants)	1	
Printing of the board and board labels, as depicted in Figure 1	3	
Randomly placing the ship	4	
Checking for invalid user input (letter followed by number)	4	
Checking for invalid input (repetitive guesses of same cell, out of board, etc)	4	
Updating the board cells based on user input	4	
Detecting end of the game	4	
Displaying a correct score after a win	2	
Terminate the game after a win	2	
Usage of comments and readability of code (style, variable naming, etc.)	2	

Comments from the Grader: