**Due Date:** Tuesday, March 6th, 2012, 10:00 pm

**Battleship**

In this assignment, you'll be writing the game Battleship. If you're unfamiliar with Battleship, see [here](http://en.wikipedia.org/wiki/Battleship_%28game%29) and [here](http://boardgames.about.com/od/battleship/a/Rules-of-Battleship.htm). You can play the game online [here](http://www.webbattleship.com/).

**Starter code**

For this assignment, we are giving you some [starter code](http://www.cdf.toronto.edu/%7Ecsc108h/winter/assignments/a2/starter/a2.py) to which you'll need to add the functions described below. Do not modify any of the starter code, including the values of the constants; doing so may cause the test cases to fail. Some starter code functions depend on functions you will write, so they won't run properly until you've implemented your A2 functions.

**Constants**

Constants (fixed values) are variables whose values do not change once assigned. A different naming convention (uppercase pothole) is used to signify constants, so that programmers know not to change their values. For example, in the Battleship starter code, the constant BOARD\_SIZE is assigned the value 10 at the beginning of the module and the value of BOARD\_SIZE should never change. When writing your code, if you need to refer to the size of the board, you should use BOARD\_SIZE, rather than 10. The same goes for the other constant values. They are globals, so they can be accessed from anywhere in the module.

Using constants simplifies code modifications and improves readability. If we later decide to use a different sized board, we would only have to change the size in one place (the BOARD\_SIZE assignment statement), rather than throughout the program. Note: do not change the values of the A2 constants, as the test cases may depend on them having the same values as the starter code.

**An overview of the design:**

* The module contains several constants:
  + BOARD\_SIZE: the width and height of the board
  + HIDDEN: the symbol representing a hidden cell
  + VACANT: the symbol representing a vacant cell
  + SHIPS: a list of strs containing the symbols representing each ship
  + SIZES: a list of ints containing the size of each ship from the corresponding element of SHIPS
* Each player will have two lists of lists of strs used to represent their boards:
  + The "view board" is what the player's opponent will see and it is initially made up entirely of HIDDEN symbols.
  + The "symbol board" is what is initially entirely hidden: the symbols for the ships from SHIPS and the VACANT cells.

As the game progresses, the "view board" is updated with the values from the "symbol board" based on the opponent's guesses.

* Each player also has a "hits list" for their ships, which is a list of ints that contains the number of hits remaining to sink the ship in the corresponding element of SHIPS. A "hits list" initially has the same values as SIZES and the game ends when one player's "hits list" contains all 0s.
* To play the game, two different main functions have been provided. One involves only a single player and is included for testing purposes (we don't typically play against ourselves!). The other allows a human player to play against the computer. If you complete the assignment and are up for another challenge, you could add a two (human) player main or make other enhancements (this is just for fun, not for marks!).

To gain a better understanding of how the functions you'll write are to be used, read through the starter code to see where the functions are called and which values are passed as arguments. Reading and understanding the starter code is an important part of completing this assignment.

**Terminology: row, column, and cell**

In this handout and in the starter code, we refer to a board's "cell", where a board is a list of lists of strs. A cell is a single board location as specified by its row and column. For example, board[3][5] refers to the cell in row 3 and column 5. The first row of a board is row 0 and the first column is column 0.

**What to do**

In the starter code file a2.py, complete the following function definitions. Include a docstring comment for each.

|  |  |
| --- | --- |
| **Function name: (Parameter types) -> Return type** | **Description** |
| in\_bounds: (int, int) -> bool | Return True if both ints are between 0 (inclusive) and BOARD\_SIZE (non-inclusive), and return False otherwise. |
| is\_win: (list of ints) -> bool | Return True if all elements of the list are 0 or if the list is empty, and False otherwise. |
| get\_view\_board: () -> list of lists of strs | Return a board with BOARD\_SIZE rows and BOARD\_SIZE columns, where each cell contains the HIDDEN symbol. |
| get\_symbol\_board: (str) -> list of lists of strs | Return the board contained in the file with the given name. (See the example board file [sample\_board.txt](http://www.cdf.toronto.edu/%7Ecsc108h/winter/assignments/a2/starter/sample_board.txt).) |
| is\_revealed: (int, int, list of list of strs) -> bool | The first two ints are a row and column, and the last parameter is a view board. Return True if cell at that row and column is revealed (not hidden), and return False otherwise. |
|  |  |
| is\_occupied: (int, int, int, int, list of list of strs) -> bool | The first two ints are a row and column, and the third and fourth ints are a another row and column. The last parameter is a board. Return True if the path from the first row and column cell to the second row and column cell, including those two points, is not completely vacant, and return False otherwise. You may assume that the rows and columns given will form either a horizontal or vertical path, not diagonal. |
|  |  |
| is\_hit: (int, int, list of lists of strs, list of ints) -> int | The first and second parameters are a row and column, the third is a board and the fourth is a hits list for that board. If the cell at that row and column of the board contains a ship, decrease the number of hits remaining to sink that ship by one at the appropriate position of the hits list, and return the number of hits remaining. If the cell does not contain a ship, return -1. |
| update\_board: (int, int, list of lists of strs, list of lists of strs) -> NoneType | The two int parameters represent the row and column of a cell. Set the element of the cell at that location in the first list to value of the corresponding element from the second list. |
| get\_num\_moves: (list of lists of strs) -> int | The parameter is a view board. Return the number of moves made so far for the board, based on the number of non-HIDDEN elements. |

**No Input or Output!**

Your a2.py file should contain the starter code, plus the function definitions specified above. The function definitions that *you* write must not include any print statements or calls to media.choose\_file or raw\_input.

**How you should verify your code**

One way to test your code is to play the game using one of the main functions given in the starter code. However, you should also test each function individually by writing code to verify your a2 functions in an if name == '\_\_main\_\_:' block in a2.py. All verification code, including any calls to the starter code functions, should be entirely within the main block and not anywhere else in a2.py.

**Type checks**

We are providing a type-check module that can be used to test whether your functions have the correct parameter and return types. To use the type checks, place [a2\_type\_checks.py](http://www.cdf.toronto.edu/%7Ecsc108h/winter/assignments/a2/a2_type_checks.py) in the **same** directory as your a2.py and Run it.

**If the type-checks pass:** Then your function parameters and return types match the assignment specification. **This does not mean that your code works correctly in all situations.** We will do a thorough job of testing your code once you hand it in, so be sure to thoroughly test your code yourself before submitting.

**If the type-checks fail:** Look carefully at the message provided. One or more of your parameter or return types does not match the assignment specification. Fix your code and re-run the tests. Make sure the tests pass before submitting.

**Our Tests**

For the exercises, your code was pre-marked, so you got test results before the deadline. For the assignment, you will not get test results until after the deadline has passed. Therefore, it is up to you to verify your code to determine whether or not it is working. Once the deadline has passed, we will run a set of tests on your submission.

**Marking**

These are the aspects of your work that may be marked for A2:

* **Correctness:** Your functions should perform as specified. Correctness, as measured by our tests, will count for the largest single portion of your marks.
* **Formatting style:** Make sure that you follow [Python style guidelines](http://www.cdf.toronto.edu/%7Ecsc108h/winter/assignments/a2/rules.shtml) that we have introduced!
* **Programming style:** Your variable names should be meaningful and your code as simple and clear as possible. Where possible, you should avoid duplicate code by calling other functions within the module as helpers. You should use constants rather than "magic" values (e.g., use BOARD\_SIZE, not 10). Your functions should be written so that they would work properly, even if different values are used for the constants.
* **Commenting**: We want to see great docstrings and internal comments. Follow these [guidelines](http://www.cdf.toronto.edu/%7Ecsc108h/winter/assignments/a2/rules.shtml).
* **Organization**: Whenever you find yourself duplicating code, think about whether that code can be stored in a function and called multiple times.

**What to Hand In**

**The very last thing you do before submitting should be to run the type-check module one last time.** Otherwise, you could make a small error in your final changes before submitting that causes your code to receive zero for correctness.

Submit a2.py according to the instructions on the course website. Remember that spelling of filenames, including case, counts: your file must be named exactly as above. Remember also that your submission will be marked on CDF, so you should make sure it works on CDF.