CS111 Spring 2014

Second Programming Project

March 11, 2014

Your assignment is to write a Python program that plays the game of Battleship with the user. Your program is due **Monday, April 4th, at Midnight**. You will be asked to submit two files.

1. Your file of Python code. Name it **ships.py**.
2. A *trace file* showing how your code works on a particular set of inputs.

See the end of this handout for more information about what to submit.

Battleship Battleship is a guessing game played with pencil and paper. In our version, the computer creates a a 10 by 10 grid (rows labelled by numbers and columns labelled by letters), and secretly places five ships on the grid.

The user then guesses the location of the ships, for example by typing **5 B**. The computer responds with **miss** if there is no ship at that location and with **hit** if there is a ship at that location. If the user has managed to hit all parts of a given ship (they extend over several grid squares), then the computer also reports **sunk**. Visit **wikipedia: Battleship_(game)** for more information about how to play this game.

The ships have the following names.

name	squares
patrol boat	2
destroyer	3
submarine	4
battleship	5
aircraft carrier	6

Note: You don't have to call your game "Battleships." You can put down 5 farm animals (mouse, chicken, pig, goat, cow) and have the user try to "tag" and "pen" them or whatever, as long as the basic parameters of the game are met.

After each guess, the computer displays what the user knows about the grid so far (that is, which squares have been tried, labelled **hit** and **miss**. It also reports how many guesses have been used, and how many ships have been sunk. Figure 1 shows an example of what the grid might look like half-way through a game:

```
a b c d e f g h i j
~ ~ ~ ~ M ~ ~ ~ ~ ~ 1
~ ~ H ~ ~ ~ M ~ ~ ~ 2
~ ~ H ~ ~ ~ ~ ~ ~ ~ 3
~ ~ ~ ~ ~ S S S ~ ~ 4
~ ~ ~ ~ ~ ~ ~ ~ ~ ~ 5
~ ~ M ~ ~ ~ ~ ~ ~ ~ 6
~ ~ ~ ~ ~ ~ M ~ M ~ 7
~ H H H ~ ~ ~ ~ ~ ~ 9
~ ~ ~ ~ ~ ~ ~ ~ S S 10

Guesses remaining: 35
```

Figure 1: A typical Battleship Board mid-game

This figure shows that the user has already made 15 guesses. So far the submarine and the destroyer have been sunk (in rows 4 and 10), and two others have been hit but not yet sunk. The user has missed 5 times.

The object of the game is for the user to sink all five ships before running out of guesses (I suggest you use 50 guesses, but this is flexible). Whichever happens first, the computer stops the game and tells the user about it.

Here are some details about how to write this program:

Placing the ships on the board. The first problem to tackle is how to place the ships on the board when the game begins. Here are some rules.

- To represent the board you probably want to use a list of strings (one string per row), or a list of lists of strings (one string per square). You will need to decide which string values represent which kinds of states on the board.
- The five ships take up different numbers of squares and each is always laid out in a straight line.
- You can choose whether any given ship is placed horizontally or vertically.
- Two different ships should not overlap, of course. Nor should they be placed next to one another – that is, no two ships should be in adjacent squares. Put another way, each ship should be surrounded by a ring of water at least one square deep.
- You can use a random method to place the ships, or a semi-random method, or whatever you like. If you use a completely random method the main problem will be

checking that your ships do not run into one another when you place them on the grid.

- Include a cheat function so the professors can test your program without actually guessing. Ask for the user's name, and if the user types "cheat" the ships should be placed horizontally in the upper left hand corner like this (c=carrier, b=battleship, s=submarine, d=destroyer, p=patrol)

```
c c c c c c ~ ~ ~ ~
b b b b b ~ ~ ~ ~ ~
s s s s ~ ~ ~ ~ ~ ~
d d d ~ ~ ~ ~ ~ ~ ~
p p ~ ~ ~ ~ ~ ~ ~
~ ~ ~ ~ ~ ~ ~ ~ ~ ~
...
```

  That way the cheater knows where to find them. (They do not have to be displayed, however.)

Running the game Once the ships are placed, the program iterates: it prompts the user for a guess, checks the guess against the board (updating the board) and then reports what happened (hit, miss, sunk) and how many guesses are left. Here are some thoughts:

- At the beginning, the board is all ocean. You can choose how to represent the ocean. As the user plays the games, the grid squares should be filled in with H, M, or S, as appropriate. A partially-hit ship has H's on its parts, but when it is sunk all the H's should turn to S's.
- How do you tell when a ship is sunk, as opposed to just hit? You might want an extra table of information to keep track of this, or else invent a way to represent this information directly on your board.
- The internal representation of the board has more information (like the location of the ships) than you want to print out. How will you translate the internal representation into the version that is printed?
- For good style, your program should check that the user has provided valid inputs (row numbers between 1 and 10, column numbers between a and j), and if not, invite her to try again.
- If you like, invite the user to play again, and start the game over. If you like, ask if the user wants to see the rules, and print the rules.

- The next lab will include an introduction to turtle graphics, which allows your program to draw pictures. If you like, you can use the turtle to draw the board instead of printing it out.

What to turn in. As usual point your browser to `www.cs.amherst.edu/Submit`, find the Program 2 submit page, and turn in the following two files.

1. The program **ships.py**, of course.
2. A file named **prog2script.py** that is a transcript record of you (as user) interacting with your program.
3. First, your transcript should show your program running in cheat mode: make two wrong guesses, then make a guess to hit each ship once (guessing in row a). Then sink the patrol boat. Second, your transcipt should include inputs that show off what kind of error checks your program has.
4. It is ok to submit multiple times, but you must include **both** the program and the script files each time you submit. We only look at files from the latest submit timestamp.

Note: Do not try to edit the **prog2script.py** file, for example to clean up typos and errors that you might have created as user. You will not be penalized for being a clumsy user. But, if you edit the file you may create mis-matches between what the script says your program does and what your program actually does. A professor who notices such a mismatch will wonder if you're trying to fake the output to get a better grade. You of course do not want to alarm the professor and waste his or her time in that way.