CS 100 Programming I
Project 3

Revision Date: October 2, 2015

Preamble

You may develop your code anywhere, but you must ensure it runs correctly on a Linux distribution before submission.

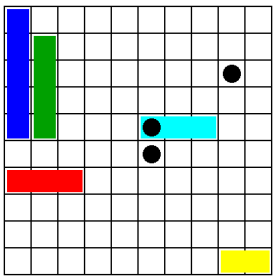
Battleship!

> "You sunk my battleship!" - Annoying TV Kid

Battleship is a classic Milton-Bradley game, where two players, in secret, position their navy on a grid. Each ship in the navy covers a contiguous horizontal or vertical section of grid cells. The type of ship determines the number of grid cells covered. The players, in turn, lob salvos at each other's navy by calling out grid locations. Should a salvo coincide with a grid cell covered by a ship location, the attacked player calls out "hit". If the salvo does not coincide with a cell covered by a ship, the attacked player calls out "miss". If all the cells covered by a single ship are hit, the attacked player calls out "sunk".

The first player to sink all of the opponent's ship wins.

Here is an example of one player's grid after three salvos by the opponent:



The salvo to grid location (2,8) was a miss, the one to location (4,5) was a hit, and the one to (5,5) was another miss.

The grid also illustrates a typical navy: a carrier in blue, a battleship in green, a destroyer in red, a submarine in aqua, and a patrol torpedo boat in yellow.

Your task is to build a battleship-playing program that will outsmart your classmates' programs. May the best program win!

Program-to-program communication

The two competing programs will be designated player A or player B. When each program starts up, it will position its navy and write out ship positions to a file given as a command-line argument. Player A will start the competition by appending a grid location to the file *A.salvos*. Player B will then read the *A.salvos* file and append the response, either MISS, HIT, SUNK, or DEFEATED, to the file *B.responses*. Player B will then append a grid location to the file *B.salvos*. Player A will read *B.salvos* and append a response to *A.responses*. Player A then reads *B.responses* to guide its next salvo. This continues until a program has its last ship sunk, at which time, the program should print the message DEFEATED and quit.

If a program is deemed to have purposefully corrupted the salvos or response files or attempts to read the opponent's ship position file, its authors will receive, at a minimum, a zero for the assignment and the incident will be filed under the UA and CS 100 academic misconduct policy for dismissal from the course with a failing grade.

File formats

The ships position file will consist of a number of entries. Each entry is composed of a ship designator followed by row and column indices. The legal ship designators are the following letters:

designator	ship type	number of cells
C	carrier	5
B	battleship	4
D	destroyer	3
S	submarine	3
P	patrol torpedo boat	2

The grid is made up of 10 rows and 10 columns. The top row of the grid is row 0 and the leftmost column of the grid is column 0.

Each entry in the salvos file is composed of row and column indices. For the prior example, the salvos file would look like:

```
2 8
4 5
5 5
```

The corresponding responses file would look like:

```
MISS
HIT
MISS
```

With respect to the responses file, if a hit results in a sunk ship, only the response SUNK is appended to the file. If a hit results in the last remaining ship being sunk, only the response DEFEATED is appended to the file.

NOTICE: all files are free format. It would be perfectly OK to try to cause your opponents program to fail by adding extraneous whitespace (within reason) to an addition to the salvos or responses file. For example, it would be legitimate to append the salvo (6,8) to the salvos file as:

```
2 8
4 5
5 5

    6

8
```

All that is required is that the file is appended and the row comes before the column.

Remember to close all files after appending to them!

Slowing down the action

Your program should pause at the very beginning if it is player B.

Your program should also pause after each addition to the salvos files, waiting until you tell it to continue. This will allow the other program time to read the salvos file and append a response and a new salvo. At each pause, your program should print out the status of your ships and the status of your salvos. For example, the status of your ships (on the left) and salvos (on the right) might look like:

```
    ships                   salvos
    - - - - - - - x P -     - x o - - - - - - -
    - C - - - - - - - -     - x o - - - - - - -
    - C - - - - - - - -     - o o - - - - - - -
    - C - x x B B - - -     - - - - - - - - - -
    - C - - - - - - - -     - - - - x - - * - -
    - C - - D - - - - -     - - - - - - - x - -
    - - - - D - - - - -     - - - - - - - o - -
    - - - - D - - - - -     - - x - - - - - - -
    - - - - S S S - - -     - - - - - - - - - -
    - - - - - - - - - -     - - - - - - - - - -
round: 22
```

Here, the *BCDPS* letters stand for the ships and the *x*s and *o*s refer to hits and misses, respectively. The * indicates the latest salvo. After the boards are printed, the number of the round should be printed. Player A's rounds are 1, 3, 5, 7, ... while Player B's rounds are 2, 4, 6, 8, ... The round numbers are useful for keeping track of whose turn it is.

Reading and writing to files

To read the *salvos* and *responses* files, you may use the *Scanner* class from previous assignments. There is a new version so you should delete your old *scanner.py* and get the new one:

```
wget troll.cs.ua.edu/cs100/python/projects/scanner.py
```

To write to a file, you need to read Chapter 8 of your textbook. Make sure you open the *salvos* and *responses* files in append mode so you don't wipe out what's already there.

Command line arguments

Your program shall be started with one command line argument. The argument is the letter *A* or the letter *B*, designating player A or player B. Here is an example:

```
python3  battleship.py  A
```

Stepwise Refinement

The stepwise refinement methodology of the previous programming assignment exhibited a *top-down* approach. In a to-down approach, the main function is sequentially refined until the project is finished. For this project, you should take a *bottom-up* approach, in which you start out by writing the most basic functions and then write higher-level functions that call these basic functions, and so on.

Level 0 Define two functions, one that returns the last response in the opponent's response file and one that returns the last location in the opponent's salvos file. Test your functions thoroughly so that you are convinced that they are working. Do this thorough testing for all functions you define.

Level 1 Define two functions, one that appends a response (HIT, MISS, or SUNK) to your response file and one that appends a location (row and column) to your salvos file.

Level 2 Define a function that creates a board. A board can be represented as a list of 10 lists, each 10 items long. You will eventually use this function to create two boards. One board will represent your placement of ships. The other will record your hits and misses when you salvo your opponent.

Level 3 Define a function that, given a board, a row, a column, a size, and an orientation (horizontal or vertical), tells whether or not a ship of the given size can be placed at the given spot with the given orientation without overlapping a previously placed ship or extending beyond the edge of the board. You will need to be able to tell the difference between an empty spot on the board and an occupied spot.

Level 4 Define a function that places a ship on a board, perhaps randomly. The orientation (horizontal or vertical) can be generated randomly as well. You should pass in the type of ship: C, B, D, S, or P; and the size of the ship: 5, 4, 3, 3, or 2.

Level 5 Define a function that prints out a representation of both boards This function should print out the number of the round as well. After printing the boards and round number, the round number should be incremented by 2. Have the round number be held by a global variable.

Level 6 Define a function that, given a salvo location and a board, determines whether the salvo was a hit or a miss. If the salvo was a hit, it should update the number of hits for the ship that was struck and see if the ship was sunk. The function should return HIT, MISS, SUNK, or DEFEATED (if the last ship was sunk).

Level 7 Define a function that, given a board somehow marked as to whether your previous salvos were hits or misses, determines your next salvo location.

Level 8 Define two functions, one for player A and one for Player B. The function for player A should do the following in a loop:

- generate the salvo location
- append the salvo to your salvos file
- print both boards and the number of the round
- pause

- read the response from the opponent's responses file
- update the opponent's board
- read the opponent's salvo location
- determine your response to the opponent's salvo
- update your board
- append your response to your responses file

The function for player B should do the following in a loop:

- read the opponent's salvo location
- determine your response to the opponent's salvo
- update your board
- append your response to your responses file
- generate the salvo location
- append the salvo to your salvos file
- print both boards and the number of the round
- pause
- read the response from the oppenent's responses file
- update the opponent's board

Level 9 Tie all these functions into the finished program. You may need other functions to do this.

Level 9 - Optional Improve the function that determines the next salvo by preferring locations that are next to a previous hit

Level 10 - Optional Improve the function that determines the next salvo by preferring locations that are colinear with two previous hits

Note, for each function you write at each level, you should use top-down stepwise refinement. That is to say, start with the simplest version of the function and successively add capability to the function.

Submission Instructions

Change to the directory containing your assignment and run the command:

```
submit cs100 xxxx project3
```

Replace `xxxx` with your instructor's **nixie** login name.