CS396: Principles of Languages – Fall 2017

Homework	Points	Announced	Due
#2	8	Oct-09	Oct-30

Introduction

The purpose of this homework is to have more functional programming exercises.

Instructions

Read the following instructions carefully **before** working on this assignment.

1. This is an **individual** assignment. You may discuss ideas, ask questions or explain things to each other. Nevertheless, you should solve the assignments independently.
2. Submissions via email will not be accepted. The homework should be submitted via BBLearn by the due date.
3. You should submit your own work. Material brought from elsewhere (e.g. the Internet[1], a classmate, submission at a previous offering…) is not acceptable.
4. Submit a single DrRacket file (.rkt)[2] that includes the supplied code and your own code.
5. Add comments to your code.
6. A program with syntax errors will earn **zero** points.
7. Scheme was polluted with imperative programming constructs. You must avoid using these:
    a. Avoid explicit looping and use recursion instead.
    b. Avoid using sequencing constructs such as begin.
    c. Use only *lists* to structure your data.
8. You can use these features:
    a. Defining/using global variables in your code.
    b. Set functions such as set!, set-car! and set-cdr!
    c. The display function.
    d. Functions defined in libraries provided by Dr. Racket and/or cs396-lib.rkt

The Battleship Game[3]

In this homework, we will implement the battleship game using Scheme. Battleship is a guessing game for two players. It is played on ruled grids (paper or board) on which the players' fleets of ships (including battleships) are marked. You have a board that flips open sort of like a laptop (see image).

On the flat part, you place some ships without letting your opponent see of course. Your opponent does the same on the other side. Then you just take turns calling out board coordinates, things like "b7" and "d3". Your opponent tells you whether they are a "hit" or a "miss", and you mark your shots on the vertical board so you know where you've shot. Ships have different hit values depending on their size: it takes two hits to sink a little destroyer, a battleship takes three, an aircraft carrier four. If a player has a ship that has taken all its hits, that ship must be announced as sunk. The game is over when someone has no more ships above water.

[1] Unless explicitly told to do so.
[2] No zip files. No .doc files. No .docx files. No .pdf files
[3] This assignment was designed by Dr. Eck Doerry

Like all board games, the data structure is: **a list with nested lists of "rows" inside it**, perfect for practicing recursive list manipulation! It's nothing complex, and we made a few compromises to keep things ever simpler, e.g., ships can be placed only horizontally on the board (vs. horizontally or vertically, like in the picture)

The game is already mostly implemented (see attached Battleship.rkt) Your job is to **write one key function in the game, the one that actually drops the bombs**, plus any helper functions your need behind the scenes.

Your task is to **complete the body of the *bomb* function, plus any other helper functions** you want to add to help you get the job done. The *bomb* function header is already defined in the attached code. **Don't change it!** The provided code needs exactly this function signature to work! **Here are some details to consider in implementing this function**:

- The parameters are straightforward: bomb just takes three arguments: a row (int), a column (int), and a single character, which is either 'P or 'C. If it's P, it is the players board that is being bombed; if it's C, then it's the computer's ships under attack. You'll use this inside the bomb function to determine which "ocean" (board) to work on.
- Your bomb function (or its helpers) must figure out if the bombed location resulted in a hit or a miss. This should be reported clearly, as you can see in the output (see BattleshipOutput.txt).
- You must also mark the hits/misses on the appropriate boards. This is simple: as you can see in the sample, ships are represented on the board by sequences of A, B, C, or D -- namely, capital letters. When a ship gets hit by a bomb, that capital letter is changed to a lower-case letter. Misses are marked as lowercase o's.
- You have to notice and mark sunk ships. Once a ship has been filled with hits, it must be reported clearly (see BattleshipOutput.txt) as sunk. It is also marked as sunk by replacing the (now) lowercase letters of the ship with lowercase s's (see BattleshipOutput.txt). Sunk ships should be removed from that player's ship list.
- Your bomb function (or its helpers) must also monitor the game for winners. This is pretty easy: when some players ship list goes to zero (all ships sunk and deleted), the game reports a winner. See sample output (see BattleshipOutput.txt).
- The bomb function just returns any old thing. It's irrelevant. The provided one returns #t if it finished successfully, and displays "invalid input" if not, e.g. if garbage parameters were passed in. The point is that this function (or its helpers) work by updating (**through set!**) the global game state (boards and ship lists), so it doesn't really matter what it returns.
- You need to look carefully at a few pieces of existing code. The provided code is irrelevant to you for the most part (except maybe to study and pick up some insights), but a few bits are important to know. In particular, note the key "data" in this game, the two boards (cOcean and pOcean), as well as the ship lists (cShiplist, pShiplist) defined near the top. These are the "state" of the game. Your bomb function (or one of its helpers) will need to look at these boards and, in a final step, update them (yes, using set!). Likewise, you'll want to update (i.e. delete elements from) the two ship lists as boats get sunk.

When you finish, your game should essentially function just like the provided sample output (see BattleshipOutput.txt). In addition, we may try to feed your bomb function all sorts of baloney input, so **make sure you idiot-proof that function** well. For instance, if I tell it to bomb location (99, 223), it should politely decline with a graceful error message rather than crashing horribly.

There is a helper function that you need to implement. The function is called $markchar$, and is responsible for simply placing a symbol at a designated location on a board. The complete spec and header is in the provided code. The file (BattleshipMarkchar-out.txt) shows a sample output of this function.

With best wishes
Dr. Mohamed Elwakil