CSE341 Spring '05 Assignment 1

Due Friday, April 8

Several assignments in this course will involve the once-popular game Battleship. Perhaps the game is still popular, but your TA is too old and out of it to really know.

All necessary Battleship information is in this paragraph; ask if something is unclear. Not all of this will be used in this assignment. The Battleship game takes place on a 10×10 grid representing the ocean. Ships are placed on this ocean; each ship occupies several adjacent squares arranged either horizontally or vertically. A fleet is five ships: a carrier, of 5 squares; a battleship, of 4 squares; a cruiser, of 3 squares; a submarine, of 3 squares; and a destroyer, of 2 squares.

The game involves two players. Each player sets up their fleet on their own ocean, hidden from the other player's view. Then the players take turns firing shots, trying to sink the other player's fleet. Each shot is a coordinate pair such as $(1, 2)$ (in the original game the x-coordinate was denoted by a letter, but for this homework assignment just use integer pairs). If the shot corresponds to the square of a ship, the opponent announces that a hit was scored as well as the type of the ship that was hit. Otherwise, the opponent announces a miss. If all the squares of a ship are hit, the ship is sunk and the opponent must announce that. An advanced version of the game lets players fire *salvos* of shots, which is a group of shots, one for each ship the player has left. For each salvo fired, the opposing player announces only the number of shots that hit, not which ones hit.

In the following problems, we will represent each ship by a list of squares, for example a submarine might be `[(1,2),(1,3),(1,4)]:(int*int) list`. In your solution write the argument types explicitly. For example, `fun ship_size (ships:(int*int) list) = ...`. You do *not* need to check if the input is valid (shots are inside the grid, ships are well-formed, and so on).

1. Write a function `ship_size` that takes a ship and returns the number of squares composing it. For example, `ship_size([(1,2),(1,3)])` should be 2.
2. Write a function `shot_hit` which takes a coordinate pair and a list of squares representing a ship, and returns `true` if the shot hits the ship.
3. Write a function `ships_collided` that takes two ships (lists of squares) and returns `true` if the ships intersect. Hint: use `shot_hit`.
4. Write a function `biggest_ship` that takes a fleet (a list of ships, `(int*int) list list)`), and returns the length of the largest ship. Hint: use `ship_size`.
5. Write a function `salvo_hits` which takes a list of shots and a fleet, and returns the number of hits. Hint: if the arguments to your function are `salvo` and `fleet`, make a recursive call on `(hd salvo)::nil` and `(tl fleet)`, and `(tl salvo)` and `fleet`.
6. Now consider a ship damage report pair `(size,hits)`, where `size` is the size of the ship and `hits` is the number of hits it has sustained. A list of these damage reports is valid if for each item, the number of hits is at most the size of the ship. Write a function `valid_damage_reports` which takes a list of damage reports and returns `true` if they are valid.
7. A ship in a damage report is sunk if the number of hits is at least the size of the ship. Write a function `ship_sunk1` that, given a list of damage reports, returns a list consisting of the damage report of one sunk ship, if such a ship exists, or an empty list otherwise. Assume the list is valid.

8. It is bad style to use a list (which can hold any number of elements) when you will always want zero or one elements. In Java, the constant `null` (not to be confused with the ML function of the same name) is useful for the "zero case"[1]. In ML, there are "options":

   - If `t` is a type, then `t option` is a type (just like `t list` is a type).
   - To make a value of type `t option`, write `NONE` (for "zero" elements), or `SOME e` (for "one" element), where `e` has type `t`.
   - The function `isSome` evaluates to `true` if and only if its argument has the form `SOME e`. The function `valOf(x : t option)` evaluates to `e` if `x` is `SOME e`. It raises an exception if `x` is `NONE`.

   Write a function `ship_sunk` that returns an option of a sunk ship damage report. That is, `ship_sunk(lst)` returns `NONE` if there are no sunk ships in `lst`, or `SOME s` if `s` is any sunk ship damage report in `lst`.

Type Summary: A correct assignment will generate the following bindings.

```
val ship_size = fn : (int * int) list -> int
val shot_hit = fn : (int * int) * (int * int) list -> bool
val ships_collided = fn : (int * int) list * (int * int) list -> bool
val biggest_ship = fn : (int * int) list list -> int
val salvo_hits = fn : (int * int) list * (int * int) list list -> int
val valid_damage_reports = fn : (int * int) list -> bool
val ship_sunk1 = fn : (int * int) list -> (int * int) list
val ship_sunk = fn : (int * int) list -> (int * int) option
```

Of course, generating these bindings does not guarantee that your solution is correct. Test your functions! My solution is 61 lines.

Assessment: Your solutions should be

- correct,
- in good style, including indentation and line breaks, and
- written using the features presented in class. In particular, you should not use references or arrays, and use the =, > and < operators only to compare `int` or `string` expressions.

Syntax Errors: Small typos can lead to strange error messages. Keep a close watch on your parentheses, and note the following examples.

- `int * int list` means `int * (int list)` and not `(int*int) list`. You want the latter for a ship.
- `fun f x : t` means the result type of `f` is `t`, whereas `fun f (x : t)` means the argument type of `f` is `t`. There is no need to write in result types (and in later homeworks no need to write in argument types).
- `fun f (x t)`, `fun f (t x)`, and `fun f (t : x)` are all wrong, but the error messages suggest you are trying to do something more advanced than what you are (which is trying to write `fun f (x : t)`).

[1]But there is no way to prevent a variable from being null!