Homework 5b

Version 8.10

Programming Language #lang htdp/isl

MUST BE DONE WITH YOUR PARTNER

MUST USE CHECKED-SIGNATURES!

Due Date Thurs at 9:00pm (Week 5)

Purpose Continue working on course project

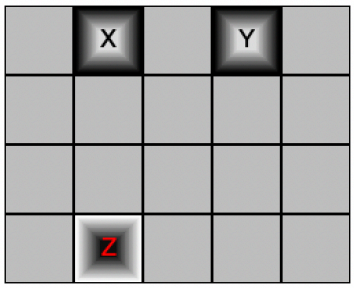
Exercises

Recall the game, Shannon, that you started working on in HW2A.

In this game, players use gates from Boolean logic (AND, OR, NOT) and to solve puzzles on space-constrained grids.

The puzzles have the following form: a grid has a fixed number of input & output wires, and a Boolean function that is the goal of the game. To play the game, players add gates & connective plates in order to implement the goal function. Given infinite space (and in 3D), this would be relatively easy, but the space and planar constraints can make it possible to create somewhat tricky puzzles.

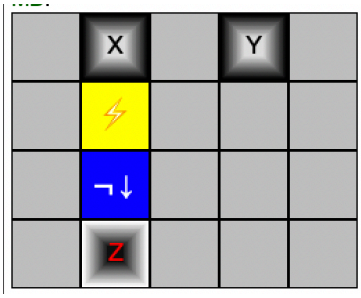
For example, a simple example grid might be:



Where the black letters are input wires, the red are output. If the function was:

```
(define (f x y)
  (not x))
```

Then the puzzle could be solved by connecting the input to output with conductive plates (which allow boolean values to flow across them in any *non-diagonal* direction) and a single NOT gate that is oriented down, e.g., as follows:



Exercise 1 In HW2a, you designed data to represent a single "Cell" in the above board (please copy that definition into your file). In the game, you may want to represent a multi-set of Cells. One way of doing so is with a List, but lists are ordered, whereas multi-sets are not. A multi-set, for review, is an unordered collection of elements that may include duplicates (whereas a set is an unordered collection with no duplicates). In this exercise, you should design a function "cell-set-equal?" that compares two lists of "Cell"s, but using (multi-)*set* equality; i.e., the order of the "Cell"s should not matter.

Exercise 2 Next, we want you to design data to represent a "Grid" of "Cells". The grid should be two-dimensional, which means that every "Cell" can be addressed with a "Posn" (for its X and Y coordinate). You should re-use your prior definition for "Cell". There are multiple ways of designing this data, but whatever you do will impact the next several functions.

Exercise 3 Now, design a function, "grid->image", that turns a "Grid" into an image, suitable for displaying to a player of the game. You should re-use the "cell->image" function you designed previously.

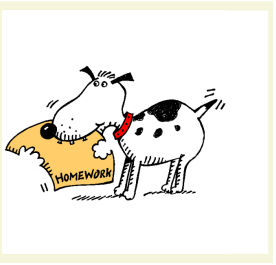
Exercise 4 Implement "get-cell", which takes a "Grid" and a "Posn" and returns the "Cell" at that position. If the "Posn" is not in the "Grid", return "#f".

Exercise 5 Implement "set-cell", which takes a "Grid", a "Posn" (i.e., a "(make-posn Number Number)"), and a new "Cell", and returns a new "Grid" where the given "Posn" has been replaced with the new "Cell". If the "Posn" is not in the "Grid", return "#f".

Exercise 6 In a "Grid", each "Cell" has up to four cardinal neighbors: i.e., "Cell"s that are immediately to the north, east, west, or south (so _not_ including diagonally to the north-east, etc). "Cell"s along the edge or corners may have fewer than four. Implement "get-neighbors", which takes a "Grid" and a "Posn" and returns a list of the cardinal neighbors of the "Cell" at the given "Posn". If the "Posn" is not in the "Grid", return "#f".

Homework 9a

Version 8.10



Programming Language #lang htdp/isl+

MUST BE DONE WITH YOUR PARTNER

MUST USE CHECKED-SIGNATURES!

Due Date Tues at 9:00pm (Week 9)

Purpose Revise Shannon, and turn it into a game.

Exercises

Exercise 1 All of your games were at least several hundred lines of code; some were quite a bit more. Some of them worked perfectly, others partially worked. Some had extensive tests, others very few. This exercise involve several parts, all aimed towards improving your existing code:

A. Incorporate feedback given in grading. That might have involved breaking massive functions into appropriately sized helper functions, reformatting code, etc.

B. Pay particular attention to testing. If your parse-cell and parse-grid functions were not useful for testing, please extend them so that you can use them in your tests (e.g., you should be able to use them for both the inputs & outputs of your functions): they should make your tests much more concise and easy to read. If they don't, pick a different format to parse from (you do *not* have to change your grid definition: parsing converts, so can handle a different representation).

C. If your code was not fully functional to the point of supporting Homework 6b, please fix whatever outstanding bugs that it has. If it didn't have enough tests, add them.

D. Organize the file to collect together data definitions, parsing functions, grid functions, etc. Using large comments like:

```
;;;;;;;;;;;;;;;;;;
;; SECTION NAME ;;
;;;;;;;;;;;;;;;;;;
```

can make scrolling through the file much easier.

E. If there are any other changes you think could make your code easier to read or understand, please make them (note: aside from comments like in (D) and purpose statements, we do not want you to add other comments—if you find yourself wanting to explain a function, it may indicate that the function is doing too much: splitting it into smaller functions is much better, as they, unlike comments, can never become out of sync with the code).

This should result in significant changes to your code: we will be grading it based on the difference between what you submitted in 6B and here. If you make only a few cosmetic changes, very likely, you will get a very low score.

At the end of it, you should be happy if we decide to project your code in front of the whole class and have you describe how it works.

PLEASE INCLUDE A (CONCISE) SUMMARY OF CHANGES AT THE TOP OF THE FILE.

Exercise 2 Your next task is to turn Shannon into a game. First, update your game world state so that it supports a *goal*, which should be a set of inputs/outputs. e.g., you might want to be able to represent

```
X:+,Y:+ -> Z:-
X:+,Y:- -> Z:+
X:-,Y:+ -> Z:+
X:-,Y:- -> Z:+
```

This is a table version of the function (lambda (X Y) (not (and X Y))), assuming the only values you have are booleans and "+" (positive) corresponds to #true and "-" (negative) corresponds to "false". While we show that one example as a function, you need to represent them in this "relational" form, since you need to support *multiple* outputs (otherwise, with only booleans, there are limits in how interesting things get), and you need not include *all* combinations of inputs. e.g., you should be able to equally represent this as a set of goals:

```
R:+,S:+ -> T:-,U:+
R:-,S:- -> T:+,U:+
```

You can assume that whoever constructs the goals will do so with inputs/outputs that exist in the grid.

These are *goals* because each corresponds to a possible simulation of the circuit: i.e., consider the first case of the first example. It indicates that if we set the inputs X and Y to +, then after running the circuit, the output Z should have charge -. In addition to the goal itself, you probably want to record whether or not the goal has been simulated and what the result was.

Exercise 3 You should make a new to-draw handler that, in addition to showing the grid, also shows the goals and whether they have been satisfied. We would suggest that you don't *edit* your old draw handler, but rather, make your new one treat it as a helper (depending on how you updated your world state, this is hopefully easy!).

Exercise 4 Now that you have a goal, you should create a new on-tick handler that proceeds through the goals one at a time, running the simulation until it has stopped changing and recording whether the particular goal was sucessful or not. It is fine if short-circuits cause the game to exit out. (As with the to-draw, we would suggest that you have your new handler call the old one as a helper.)

Exercise 5 Finally, update your main function, and be sure you have some visual indication, once all the goals have been run, whether the game was won or lost.

Exercise 6 In order to play the game, a player must configure an initial state of the grid with gates that they believe will satisfy all the goals. Please make two "puzzles": states with just inputs, outputs, and the set of goals. You should use your parse-grid function, and be sure your goals are readable (make a parse-goal function if necessary) so that a player can understand what they are supposed to fill in. You should then also provide solutions (filled in versions of the states) to each of your puzzles. Your puzzles should not be trivial, but you do not have to make them too large (roughly 10x10 grids are probably about right, with a few inputs and outputs).