# Milestone 7

due at midnight on Sun May 7 (125 points)

On 4/26, I committed a partial solution to M6 to the repository, in the `minesweeper/` directory. It consists of `sweeper.py`, which implements the basic rules of the game, and contains a `__main__` segment to run it with a random player. And the other module is `constraints.py`, which partly implements our constraint solving over Boolean variables representing the coordinates.

For this milestone, I'd like you to take it a few steps further. Here are some options: do one or more:

1. Implement 'prettier' output of the board in the `render` function of the `Minefield` class. Right now it shows an uninspiring matrix of integers:

```
[[ -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.]
 [ -1.   2.  -1.  -1.  -1.   2.  -1.  -1.]
 [ -1.  -1.  -1. -99.  -1.  -1.  -1.  -1.]
 [ -1.  -1.  -1.  -1.   1.  -1.  -1.   2.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.]
 [ -1.  -1.  -1.   2.   0.  -1.   0.  -1.]
 [ -1.  -1.  -1.  -1.  -1.  -1.  -1.  -1.]
 [  0.   2.  -1.  -1.  -1.   0.  -1.  -1.]]
```

   But that could be turned into 'fancy' ASCII graphics like this:

```
a . . . . . . . .
b . 2 . . . 2 . .
c . . . X . . . .
d . . . . 1 . . 2
e . . . . . . . .
f . . . 2 0 . 0 .
g . . . . . . . .
h 0 2 . . . 0 . .
  1 2 3 4 5 6 7 8
```

   Where we include the alphabetic/numeric labels to help us humans with naming coordinates. (Internally, they'll still be represented as a pair of zero-based integers.)

2. Implement an input function that can ask for a move from a human player. They will input something like `f2` or `h8`. To prevent accidents, you should error-check that it's a two-character string, and that the row and column are in the correct range. Then you translate it to the zero-based internal coordinates, so `f2` becomes `(5,1)` and `h8` becomes `(7,7)`. Although the size of the board is configurable in this code, your human input only needs to support up to $9 \times 9$.

3. Extend the connection between the game and the constraint solver. Instead of always choosing randomly, or just relying on human input, you can take the set of known-zero coordinates from the solver, and click on any that haven't already been cleared. You can go a long way with just that, clearing empty regions.

   Here's some code that roughly shows how I selected between random actions, human input, and partial human input (only when guessing).

   ```
   while not done:
       # Uncomment below to choose random action:
       #act = env.action_space.sample()

       # Uncomment below to have human player:
       #act = env.get_human_input()

       # Uncomment below to use constraint solver:
       if len(constraints.clearQueue) > 0:
           act = constraints.clearQueue.pop()
       else:
           act = env.get_human_input()
           # Nothing is known for sure, so we have to guess.
           #act = constraints.bestProbs.pop()
   ```

   The next step after that would be to try calculating probabilities and choosing coordinates in cases where all the known-zeros have been cleared.

4. Extend the constraint solver so it can deduce more values. I recommend continuing the test-driven development approach we began in class. If you come up with a test case that you can't figure out how to implement, you can consult me for help.