Lab 4: HyperMines

The questions below are due on Monday October 02, 2017; 10:00:00 PM.

You are not logged in.

If you are a current student, please Log In (https://6009.csail.mit.edu/fall17/labs/lab4?loginaction=redirect) for full access to the web site.
Note that this link will take you to an external site (`https://oidc.mit.edu`) to authenticate, and then you will be redirected back to this page.

Table of Contents

1) Preparation

This lab assumes you have Python 3.5 or later installed on your machine.

The following file contains code and other resources as a starting point for this lab: `lab4.zip` (https://6009.csail.mit.edu/fall17/lab_distribution.zip?path=%5B%22fall17%22%2C+%22labs%22%2C+%22lab4%22%5D)

Most of your changes should be made to `lab.py` , which you will submit at the end of this lab. Importantly, you should not add any imports to the file. Submissions for the lab are due on Monday, 2 October. You may submit portions of the lab late (see the grading page (https://6009.csail.mit.edu/fall17/grading) for more details), but the last day to submit this lab will be Friday, 6 October.

This lab is worth a total of 4 points. Your score for the lab is based on:

- answering the questions on this page (0.3 points)
- passing the test cases from `test.py` under the time limit (1.7 points), and
- a brief "checkoff" conversation with a staff member to discuss your code (2 points).

For this lab, you will only receive credit for a test case if it runs to completion in under 30 seconds on the server. Each function in your submission should also contain a docstring and at least one nontrivial doctest.

Please also review the collaboration policy (https://6009.csail.mit.edu/fall17/collaboration) before continuing.

2) Introduction

Now that you've mastered 2D mines, it's time for a small field trip. `:)` There's just a small twist. Planet *Htrae* is not very different from Earth, except that space in the *Yklim way*, *Htrae*'s star cluster, doesn't have three dimensions — at least, not always: it fluctuates between 2 and, on the worst days, 60. In fact, it's not uncommon for an Htraean to wake up flat, for example, and finish the day in 7 dimensions — only to find themselves living in three or four dimensions on the next morning. It takes a bit of time to get used to, of course.

In any case, Htraeans are pretty particular about playing *Mines*. Kids on *Htrae* always play on regular, 2D boards, but champions like to play on higher-dimensional boards, usually on as many dimensions as the surrounding space. Your code will have to support that flexibility, of course. Here's the weather advisory for the week of the tournament:

```
...VERY DIMENSIONAL IN SOUTHWEST OHADI ON YADIRF...

.AN EXITING LOW PRESSURE SYSTEM WILL INCREASE NORTHWEST DIMENSIONAL
FLUX IN AND SOUTH OF THE EKANS RIVER BASIN ON YADIRF. ESIOB IS NOW
INCLUDED IN THE ADVISORY BUT THE STRONGEST FLUX WILL BE SOUTH AND
EAST OF MOUNTAIN EMOH TOWARD THE CIMAG VALLEY.

ZDI014>030-231330-016-
UPPER ERUSAERT VALLEY-SOUTHWEST HYPERLANDS-WESTERN CIMAG VALLEY-
1022 MP TDM UHT PES 22 6102

...DIMENSION ADVISORY REMAINS IN EFFECT FROM 10 MA TO 9 MP ON
YADIRF...

* DIMENSIONAL FLUX...30 TO 35 DIMENSIONS WITH GUSTS TO 45.

* IMPACTS...CROSSFLUXES WILL MAKE FOR DIFFICULT TRAVELLING
  CONDITIONS ON LOW-DIMENSIONAL ROADS.
```

Since most of the difficulty of this lab lies in implementing recursive functions, please do not use standard library modules that use recursion behind the scenes, such as `itertools`.

3) Rules

HyperMines is the Htraean twist on *Mines*. Unlike *Mines*, *HyperMines* is played on a board with an arbitrary number of dimensions. Everything works just the same as in *Mines*, except for the fact that each cell has up to $3^n - 1$ neighbors, instead of 8.

As usual, you only need to edit `lab.py` to complete this assignment. You will need to correctly implement `nd_new_game(dims, bombs)`, `nd_dig(game, coords)`, `nd_render(game, xray)` to earn full credit for this lab. As always, **please write documentation and doctests for all new functions**.

If you're starting with a copy of your 2D Mines code, you can delete the testing functions you added for Part 3: Bug hunt. They aren't needed for this lab.

One of the implementation challenges in *HyperMines* is arbitrary-depth iteration. We include a few hints (https://6009.csail.mit.edu/fall17/labs/lab4/hints) that you may find useful as you think about which recursive helper functions would be helpful in dealing with N-dimensional arrays represented as nested lists.

4) How to test your code

We provide three scripts to test and enjoy your code:

- `python3 simpletests.py` is an interactive script that will let you select which doctests to run.
- `python3 server.py` lets you play *HyperMines* in your browser![1] (https://6009.csail.mit.edu/fall17/labs/lab4#catsoop_footnote_1) The server uses your code to compute consecutive game states.
- `python3 test.py` runs all the tests used for grading, as well as any additional test cases you put in `test.py`

5) Implementation

5.1) Game state

As before, the state of an ongoing *HyperMines* game is represented as a dictionary with four fields:

- `"dimensions"` , the board's dimensions (an arbitrary list of positive numbers in *HyperMines*)
- `"board"` , an N-dimensional array (implemented using nested lists) of integers and strings. In *HyperMines*, `game["board"][x_0][...][x_k]` is `"."` if the square with coordinate $(x_0, \dots, x_k)$ contains a bomb.
- `"mask"` , an N-dimensional array (implemeted using nested lists) of Booleans. In *HyperMines*, `game["mask"][x_0][...][x_k]` indicates whether the contents of square $(x_0, \dots, x_k)$ are visible to the player.
- `"state"` , a string containing the state of the game: `"ongoing"` if the game is in progress, `"victory"` if the game has been won, and `"defeat"` if the game has been lost. The state of a new game is *always* `"ongoing"` .

For example, the following is a valid *HyperMines* game state:

```
gameN = {'dimensions': [4, 3, 2],
         'board': [[[1, 1], ['.', 2], [2, 2]],
                   [[1, 1], [2, 2], ['.', 2]],
                   [[1, 1], [2, 2], [1, 1]],
                   [[1, '.'], [1, 1], [0, 0]]],
          'mask': [[[True, False], [False, False], [False, False]],
                  [[False, False], [True, False], [False, False]],
                  [[False, False], [True, True], [True, True]],
                  [[False, False], [True, True], [True, True]]],
          'state': 'ongoing'}
```

You may find the `dump(game)` function (included in `lab.py`) useful to print game states.

5.2) Game logic

Your task is to implement three functions: `nd_new_game(dims, bombs)` , `nd_dig(game, coords)` , and `nd_render(game, xray)` . These functions behave just like their 2D counterparts, and each of them is documented in detail in `lab.py` .

HINT: the test cases we will use to check your code in this lab are pretty complex, and so they are difficult to reason about. You will want to create some test cases of your own in `test.py` , to handle more straightforward cases that are easier to reason about. Try making a few small 1-, 2-, and 3-dimensional test cases. As always, you should be prepared to discuss any test cases you create during your checkoff.

5.3) An example game

This section runs through an example game in 3D, showing which functions are called and what they should return in each case. To help understand what happens, calls to `dump(game)` are inserted after each state-modifying step.

```
>>> from lab import *
>>> game = nd_new_game([3,3,2],[[1,2,0]])
>>> dump(game)
dimensions: (3, 3, 2)
board: [[0, 0], [1, 1], [1, 1]]
       [[0, 0], [1, 1], ['.', 1]]
       [[0, 0], [1, 1], [1, 1]]
mask:  [[False, False], [False, False], [False, False]]
       [[False, False], [False, False], [False, False]]
       [[False, False], [False, False], [False, False]]
state: ongoing
```

The player tries digging at `[2,1,0]`, which reveals 1 tile.

```
>>> nd_dig(game, [2,1,0])
1
>>> dump(game)
dimensions: (3, 3, 2)
board: [[0, 0], [1, 1], [1, 1]]
       [[0, 0], [1, 1], ['.', 1]]
       [[0, 0], [1, 1], [1, 1]]
mask:  [[False, False], [False, False], [False, False]]
       [[False, False], [False, False], [False, False]]
       [[False, False], [True, False], [False, False]]
state: ongoing
```

… then at `[0,0,0]` which reveals 11 new tiles:

```
>>> nd_dig(game, [0,0,0])
11
>>> dump(game)
dimensions: (3, 3, 2)
board: [[0, 0], [1, 1], [1, 1]]
       [[0, 0], [1, 1], ['.', 1]]
       [[0, 0], [1, 1], [1, 1]]
mask:  [[True, True], [True, True], [False, False]]
       [[True, True], [True, True], [False, False]]
       [[True, True], [True, True], [False, False]]
state: ongoing
```

Emboldened by this success, the player then makes a fatal mistake and digs at `[1,2,0]`, revealing a bomb:

```
>>> nd_dig(game, [1,2,0])
1
>>> dump(game)
dimensions: (3, 3, 2)
board: [[0, 0], [1, 1], [1, 1]]
       [[0, 0], [1, 1], ['.', 1]]
       [[0, 0], [1, 1], [1, 1]]
mask:  [[True, True], [True, True], [False, False]]
       [[True, True], [True, True], [True, False]]
       [[True, True], [True, True], [False, False]]
state: defeat
```

5.4) Check Yourself

To get a feel for working with an abitrary number of dimensions, answer the following few questions about determining cells' neighbors:

In a one-dimensional game with dimensions of `[10]`, what are the neighbors of the coordinates `[5]`? Enter a Python list of coordinates below:

In a two-dimensional game with dimensions of `[10, 20]`, what are the neighbors of the coordinates `[5, 13]`? Enter a Python list of coordinates below:

In a three-dimensional game with dimensions of `[10, 20, 3]`, what are the neighbors of the coordinates `[5, 13, 0]`? Enter a Python list of coordinates below:

Take a careful look at your results for these questions. How do the results from one question help you solve the next?

6) Code Submission

Select File `No file selected`

7) Checkoff

Once you are finished with the code, please come to a tutorial, lab session, or office hour and add yourself to the queue asking for a checkoff. **You must be ready to discuss your code and test cases in detail before asking for a checkoff.**

You should be prepared to demonstrate your code (which should be well-commented, should avoid repetition, and should make good use of helper functions). In particular, be prepared to discuss:

- Your implementation of `nd_new_game`, including any helper functions.
- Your implementation of `nd_dig`, including any helper functions.
- Your implementation of `nd_render`, including any helper functions.
- Any new test cases you added to `test.py`.

7.1) Grade

You have not yet received this checkoff. When you have completed this checkoff, you will see a grade here.

Footnotes

[1] Our visualizer only works in 2D. Above that, things get hard to visualize (though there is a reasonably nice UI here (http://gravitation3d.com/xezlec/) for playing actual HyperMines). (click to return to text) (https://6009.csail.mit.edu/fall17/labs/lab4#catsoop_footnote_ref_1)

```
\
/    /\__/\
\__=(  o_O )=
(__________)
 |_ |_ |_ |_
```