Hints for *HyperMines*

Nested iteration

One of the difficulties in *HyperMines* is arbitrary-depth iteration. In *Mines*, you could write the following:

```
for r in range(nrows):
    for c in range(ncols):
        if (some condition on r, c):
            # Do something with (r, c)
```

In 3 dimensions, you could imagine generalizing it to the following:

```
for x in range(width):
    for y in range(height):
        for z in range(depth):
            if (some condition on x, y, z):
                # Do something with (x, y, z)
```

But this won't work for *HyperMines* (do you see why?). Instead, can you write a recursive function?

Neighbors

In line with the previous tip, how can you write an `nd_neighbors` function that enumerates all neighbors of a given point? The following auxiliary function template might be useful:

```
def nd_product(sequences):
    """Produce the Cartesian product of sequences.

    Arguments:
       sequences (list): Sequences to compute the product of

    Returns:
       A list of tuples

    >>> nd_product(((1, 2, 3), ("a", "b")))
    [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b'), (3, 'a'), (3, 'b')]
    """
```

Once you have the `nd_product` function, how can you use it to enumerate all neighbors of a point? It could be helpful to consider the 2-dimensional case first, then the 3-dimensional one, etc.

Suggested functions

Here is a list of useful auxilliary function that the reference solution uses. If you use any of them, remember to add your own doctests!

```
def nd_get(nd_array, coords):
    """Get element at coords in nd_array.

    Arguments:
       nd_array (list): N-dimensional input array
       coords (tuple): Coordinates of interest

    Returns:
       An array element
    """
```

```
def nd_set(nd_array, coords, value):
    """Set element at coords in nd_array.

    Arguments:
       nd_array (list): N-dimensional input array
       coords (tuple): Coordinates of interest
       value: Value to put at coords
    """
```

```
def nd_neighbors(game, coords):
    """Produce all neighbors of coords in game.

    Arguments:
       game (dict): Game state
       coords (tuple): Reference point

    Returns:
       An iterable of coordinates

    >>> game = {"dimensions": [2, 4, 2],
    ...         "board": [[[3, '.'], [3, 3], [1, 1], [0, 0]],
    ...                   [['.', 3], [3, '.'], [1, 1], [0, 0]]],
    ...         "mask": [[[False, False], [False, True], [False, False], [False, False]
    ...                  [[False, False], [False, False], [False, False], [False, False
    >>> sorted(nd_neighbors(game, (1, 2, 0)))
    [(0, 1, 0), (0, 1, 1),
     (0, 2, 0), (0, 2, 1),
     (0, 3, 0), (0, 3, 1),
     (1, 1, 0), (1, 1, 1),
     (1, 2, 0), (1, 2, 1),
     (1, 3, 0), (1, 3, 1)]
    """
```

```
def nd_mkboard(dims, filler):
    """Create a board with dimensions dims, and fill it with filler.

    Arguments:
       dims (list): List of board dimensions
       filler (Any): Value to initialize the board with

    Returns:
       A len(dims)-dimensional array

    >>> nd_mkboard((1, 3, 2), 42)
    [[[42, 42], [42, 42], [42, 42]]]
    """
```

```
def nd_game_status(game):
    """Compute game status.

    Return one of "ongoing", "victory", or "defeat".
    """
```