6.101 Recitation 15: Mines Wrap-up **4/8/24**

This sheet is yours to keep!

Question 1: What strategy did you use when refactoring the 2d-version of minesweeper? How could we refactor the original code below?

```
def new_game_2d(nrows, ncolumns, mines):
    board = []
    for r in range(nrows):
        row = []
        for c in range(ncolumns):
            if [r, c] in mines or (r, c) in mines:
                row.append(".")
            else:
                row.append(0)
        board.append(row)
    visible = []
    for r in range(nrows):
        row = []
        for c in range(ncolumns):
            row.append(False)
        visible.append(row)
    for r in range(nrows):
        for c in range(ncolumns):
            if board[r][c] == 0:
                neighbor_mines = 0
                if 0 <= r - 1 < nrows:
                    if 0 <= c - 1 < ncolumns:
                        if board[r - 1][c - 1] == ".":
                            neighbor_mines += 1
                # ... some code that was copy / paste / modify omitted
                if 0 <= r + 1 < nrows:
                    if 0 <= c + 1 < ncolumns:
                        if board[r + 1][c + 1] == ".":
                            neighbor_mines += 1
                board[r][c] = neighbor_mines
    return {
        "dimensions": (nrows, ncolumns),
        "board": board,
        "visible": visible,
        "state": "ongoing",
    }


def dig_2d(game, row, col):
    if game["state"] == "defeat" or game["state"] == "victory":
        game["state"] = game["state"]  # keep the state the same
        return 0
```

```
43
44      if game["board"][row][col] == ".":
45          game["visible"][row][col] = True
46          game["state"] = "defeat"
47          return 1
48
49      num_revealed_mines = 0
50      num_revealed_squares = 0
51      for r in range(game["dimensions"][0]):
52          for c in range(game["dimensions"][1]):
53              if game["board"][r][c] == ".":
54                  if game["visible"][r][c] == True:
55                      num_revealed_mines += 1
56              elif game["visible"][r][c] == False:
57                  num_revealed_squares += 1
58      if num_revealed_mines != 0:
59          # if num_revealed_mines is not equal to zero, set the game state to
60          # defeat and return 0
61          game["state"] = "defeat"
62          return 0
63      if num_revealed_squares == 0:
64          game["state"] = "victory"
65          return 0
66
67      if game["visible"][row][col] != True:
68          game["visible"][row][col] = True
69          revealed = 1
70      else:
71          return 0
72
73      if game["board"][row][col] == 0:
74          nrows, ncolumns = game["dimensions"]
75          if 0 <= row - 1 < nrows:
76              if 0 <= col - 1 < ncolumns:
77                  if game["board"][row - 1][col - 1] != ".":
78                      if game["visible"][row - 1][col - 1] == False:
79                          revealed += dig_2d(game, row - 1, col - 1)
80          # ... some code that was copy / paste / modify omitted
81          if 0 <= row + 1 < nrows:
82              if 0 <= col + 1 < ncolumns:
83                  if game["board"][row + 1][col + 1] != ".":
84                      if game["visible"][row + 1][col + 1] == False:
85                          revealed += dig_2d(game, row + 1, col + 1)
86
87      num_revealed_mines = 0  # set number of mines to 0
88      num_revealed_squares = 0
89      for r in range(game["dimensions"][0]):
90          # for each r,
91          for c in range(game["dimensions"][1]):
```

```
            # for each c,
            if game["board"][r][c] == ".":
                if game["visible"][r][c] == True:
                    # if the game visible is True, and the board is '.',
                    # add 1 to mines revealed
                    num_revealed_mines += 1
            elif game["visible"][r][c] == False:
                num_revealed_squares += 1
    bad_squares = num_revealed_mines + num_revealed_squares
    if bad_squares > 0:
        game["state"] = "ongoing"
        return revealed
    else:
        game["state"] = "victory"
        return revealed


def render_2d_locations(game, all_visible=False):
    nrows, ncols = game['dimensions']
    board = [[None for i in range(ncols)] for j in range(nrows)]
    for r in range(nrows):
        for c in range(ncols):
            if all_visible or game['visible'][r][c]:
                current = board[r][c]
                if game['board'][r][c] == 0:
                    board[r][c] = ' '
                else:
                    board[r][c] = str(game['board'][r][c])
            else:
                board[r][c] = '_'
    return board


def render_2d_board(game, all_visible=False):
    nrows, ncols = game['dimensions']
    board = ''
    for r in range(nrows):
        for c in range(ncols):
            if all_visible or game['visible'][r][c]:
                current = game['board'][r][c]
                if game['board'][r][c] == 0:
                   board += ' '
                else:
                    board += str(game['board'][r][c])
            else:
                board += '_'
        board += '\n'
    return board[:-1]
```

R15 Participation Credit **Kerberos : _____@mit.edu**

Hand this sheet in at the end of recitation to get participation credit for today.

Question 2: Below is a recursive all_coords function that returns a list of tuple coordinates. Modify the code below to make this function into an efficient generator.

```
def all_coords(dimensions):
    """
    A function that generates all possible coordinates in a given board.
    """
    if len(dimensions) == 1:
        return [(x,) for x in range(dimensions[0])]

    first = all_coords(dimensions[:1])
    rest = all_coords(dimensions[1:])
    result = []
    for start in first
        for end in rest:
            result.append(start + end)
    return result
```