UDACITY

PROJECT

Creating an AI Agent to solve Sudoku

A part of the Artificial Intelligence Nanodegree Program

PROJECT REVIEW

CODE REVIEW 6

NOTES

▾ solution.py 6

```
1  assignments = []
2
3  rows = 'ABCDEFGHI'
4  cols = '123456789'
5
```

SUGGESTION

logging with default level ERROR could be added to debug the code. Logs can also help to understand the algorithms. Please have a look at this link : https://docs.python.org/3/howto/logging.html. Assert statements could be used too https://wiki.python.org/moin/UsingAssertionsEffectively

```
6
7  def cross(a, b):
8      return [s+t for s in a for t in b]
9
10 # Create unit lists for rows, columns, and squares
11 boxes = cross(rows, cols)
12 row_units = [cross(r, cols) for r in rows]
13 col_units = [cross(rows, c) for c in cols]
14 square_units = [cross(rs, cs) for rs in ('ABC', 'DEF', 'GHI') for cs in ('123', '456', '789')]
15
16 # Create diagnol unit lists
17 diag_units = [[rows[i]+cols[i] for i in range(len(rows))], [rows[::-1][i]+cols[i] for i in range(len(rows))]]
18
19 # Add diagnol units to the peers dictionary
```

AWESOME

Good job (y) Additional constraints for diagonal sudoku implemented successfully :)

```
20 unitlist = row_units + col_units + square_units + diag_units
21 units = dict((s, [u for u in unitlist if s in u]) for s in boxes)
22 peers = dict((s, set(sum(units[s],[]))-set([s])) for s in boxes)
23
24
25 def assign_value(values, box, value):
26     """
27     Please use this function to update your values dictionary!
28     Assigns a value to a given box. If it updates the board record it.
29     """
30     if values[box] == value:
31         return values
32
33     values[box] = value
34     if len(value) == 1:
35         assignments.append(values.copy())
36
37     return values
38
39
40 def naked_twins(values):
41     """Eliminate values using the naked twins strategy.
42     Args:
43         values(dict): a dictionary of the form {'box_name': '123456789', ...}
```

```
44
45      Returns:
46          the values dictionary with the naked twins eliminated from peers.
47      """
```

SUGGESTION

Its a good practice to modularize the code, like according to the logic naked_twins can be split up in two methods find_twins, eliminate twins to enhance readability.

```
48      # Create dictionary of possible naked twins
49      for unit in unitlist:
50          possible_twins = {}
51          for box in unit:
52              if (len(values[box]) == 2):
53                  value = values[box]
54                  possible_twins[value] = [box for box in unit if values[box] == value]
55          # Find actual instances of naked twins and remove values from peers in same unit
```

AWESOME

Great work providing conceptual comments in between the method where important logic is coded. its a good practice and helps demonstrating your thought process.

```
56          for x, y in possible_twins.items():
57              if (len(y) == 2):
58                  for box in unit:
59                      if (box not in y):
60                          for digit in x:
61                              values = assign_value(values, box, values[box].replace(digit, ''))
62
63      return values
64
65
66  def grid_values(grid):
67      """
68      Convert grid into a dict of {square: char} with '123456789' for empties.
69      Args:
70          grid(string) - A grid in string form.
71      Returns:
72          A grid in dictionary form
73              Keys: The boxes, e.g., 'A1'
74              Values: The value in each box, e.g., '8'. If the box has no value, then the value will be '123456789'.
75      """
76      all_digits = '123456789'
77      values = []
78
79      # Convert grid into a list of numbered character strings
80      for char in grid:
81          if char == '.':
82              values.append(all_digits)
83          elif char in all_digits:
84              values.append(char)
85
86      assert len(values) == 81
87
88      # add pairs to dictionary
89      grid_dict = dict(zip(boxes, values))
90
91      return grid_dict
92
93
94  def display(values):
95      """
96      Display the values as a 2-D grid.
97      Args:
98          values(dict): The sudoku in dictionary form
99      """
100     # Copied from 'Strategy 1' lesson utils.py
101     width = 1+max(len(values[s]) for s in boxes)
102     line = '+'.join(['-'*(width*3)]*3)
103     for r in rows:
104         print(''.join(values[r+c].center(width)+('|' if c in '36' else '')
105                       for c in cols))
106         if r in 'CF': print(line)
107
108     return
109
110
111 def eliminate(values):
112     """Eliminates values from peers of each box with a single value.
113
114     Goes through all the boxes, and whenever there is a box with a single value,
115     eliminates this value from the set of values of all its peers.
116
117     Args:
            values: Sudoku in dictionary form.
```

```
    Returns:
        Resulting Sudoku in dictionary form after eliminating values.
    """
    # Create list of solved boxes
    solved_boxes = [box for box in values.keys() if len(values[box]) == 1]

    # Eliminate solved values from peers
    for box in solved_boxes:
        digit = values[box]
        for peer in peers[box]:
            # values[peer] = values[peer].replace(digit, '')
            values = assign_value(values, peer, values[peer].replace(digit,''))

    return values


def only_choice(values):
    """Finalizes all values that are the only choice for a unit.

    Goes through all the units, and whenever there is a unit with a value
    that only fits in one box, assigns the value to this box.

    Input: Sudoku in dictionary form.
    Output: Resulting Sudoku in dictionary form after filling in only choices.
    """
    for unit in unitlist:
        for digit in '123456789':
            d_boxes = [box for box in unit if digit in values[box]]
            if len(d_boxes) == 1:
                values[d_boxes[0]] = digit

    return values


def reduce_puzzle(values):
    """Uses the 'eliminate' and 'only_choice' functions as initial strategies to solve
    the puzzle, or at least reduce the number of empty boxes.

    The function stops if the puzzle gets solved or quits if it stops making progress.

    Input: Sudoku in dictionary form.
    Output: Resulting Sudoku in dictionary form after applying updates.
    """
```

AWESOME

Good work using docstrings for methods, they help in understanding the functioning of the method.

```
    stalled = False
    while not stalled:
        # Check boxes for determined values
        solved_values_before = len([box for box in values.keys() if len(values[box]) == 1])

        # Use the Eliminate strategy
        values = eliminate(values)
        # Use the Only Choice strategy
        values = only_choice(values)
        # Check again how many boxes have a determined value
        solved_values_after = len([box for box in values.keys() if len(values[box]) == 1])
```

SUGGESTION

Although not a requirement you could call naked_twins from reduce puzzle as its the 3rd strategy to prune search space

```
        # Stop the loop if no new values added
        stalled = solved_values_before == solved_values_after
        # Sanity check, return False if there is a box with zero available values:
        if len([box for box in values.keys() if len(values[box]) == 0]):
            empty_boxes = [box for box in values.keys() if len(values[box]) == 0]
            return False

    return values


def search(values):
    """Creates a tree of possibilities and traverses it using depth-first search (DFS) until
    it finds a solution for the sudoku puzzle.
    """
    # Reduce the puzzle using the previous function
    values = reduce_puzzle(values)
    if values is False:
        return False
    if all(len(values[box]) == 1 for box in boxes):
        print("\nSOLVED!\n")
        display(values)
        return values
```

```
    # Choose one of the unsolved squares with the fewest possibilities
    count, box = min((len(values[box]), box) for box in boxes if len(values[box]) > 1)

    # Recurse tree of resulting sudokus; if one returns a value (not False), return that answer
    for digit in values[box]:
        new_board = values.copy()
        new_board[box] = digit
        attempt = search(new_board)
        if attempt:
            return attempt


def solve(grid):
    """
    Find the solution to a Sudoku grid.
    Args:
        grid(string): a string representing a sudoku grid.
            Example: '2.............62....1....7...6..8...3...9...7...6..4...4....8....52.............3'
    Returns:
        The dictionary representation of the final sudoku grid. False if no solution exists.
    """

    values = grid_values(grid)
    values = search(values)

    return values

if __name__ == '__main__':
    diag_sudoku_grid = '2.............62....1....7...6..8...3...9...7...6..4...4....8....52.............3'
    # diag_sudoku_grid = '9.1....8.8.5.7..4.2.4....6...7......5.............83.3..6......9.............'
    # diag_sudoku_grid = '8.........36......7..9.2...5...7.......457.....1...3...1....68..85...1..9....4..'
    # diag_sudoku_grid =  '...............9..97.3......1..6.5....47.8..2.....2..6.31..4......8..167.87......'
    display(solve(diag_sudoku_grid))

    try:
        from visualize import visualize_assignments
        visualize_assignments(assignments)

    except SystemExit:
        pass
    except:
        print('We could not visualize your board due to a pygame issue. Not a problem! It is not a requirement.')

```

▸ README.md

RETURN TO PATH

Student FAQ