UDACITY

PROJECT

Creating an AI Agent to solve Sudoku

A part of the Artificial Intelligence Nanodegree Program

PROJECT REVIEW

CODE REVIEW 8

NOTES

solution.py 8

```
assignments = []

```

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

```
# Box - dictionary with keys for the strings in each box, an value
# for the digit (or '.' if not) in each box
rows = 'ABCDEFGHI'
cols = '123456789'

def assign_value(values, box, value):
    """
    Please use this function to update your values dictionary!
    Assigns a value to a given box. If it updates the board record it.
    """
    values[box] = value
    if len(value) == 1:
        assignments.append(values.copy())
    return values

def cross(a, b):
    """
    Input: Given two strings a and b, i.e. cross('abc', 'def')
    Returns: returns list formed by all possible concatentations
        of a letter s in string a, with a letter t in string b.
        (i.e. cross product of elements in a and elements in b)
        i.e. ['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']
    """
    return [s+t for s in a for t in b]

# Box Labels

"""
Output:
    [
        'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9',
        'B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9',
        'C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9',
        'D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9',
        'E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8', 'E9',
        'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9',
        'G1', 'G2', 'G3', 'G4', 'G5', 'G6', 'G7', 'G8', 'G9',
        'H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7', 'H8', 'H9',
        'I1', 'I2', 'I3', 'I4', 'I5', 'I6', 'I7', 'I8', 'I9'
    ]
"""
boxes = cross(rows, cols)

```

```
# Units

"""
Output:
    [
        ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9'],
        ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8', 'B9'],
        ['C1', 'C2', 'C3', 'C4', 'C5', 'C6', 'C7', 'C8', 'C9'],
        ['D1', 'D2', 'D3', 'D4', 'D5', 'D6', 'D7', 'D8', 'D9'],
        ['E1', 'E2', 'E3', 'E4', 'E5', 'E6', 'E7', 'E8', 'E9'],
        ['F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9'],
        ['G1', 'G2', 'G3', 'G4', 'G5', 'G6', 'G7', 'G8', 'G9'],
        ['H1', 'H2', 'H3', 'H4', 'H5', 'H6', 'H7', 'H8', 'H9'],
        ['I1', 'I2', 'I3', 'I4', 'I5', 'I6', 'I7', 'I8', 'I9']
    ]

    Top row is:
        row_units[0] = ['A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9']
"""
row_units = [cross(r, cols) for r in rows]

"""
Output:
    Left-most column is:
        column_units[0] = ['A1', 'B1', 'C1', 'D1', 'E1', 'F1', 'G1', 'H1', 'I1']
"""
column_units = [cross(rows, c) for c in cols]

"""
Output:
    Top-left square is:
        square_units[0] = ['A1', 'A2', 'A3', 'B1', 'B2', 'B3', 'C1', 'C2', 'C3']
"""
square_units = [cross(rs, cs) for rs in ('ABC','DEF','GHI') for cs in ('123','456','789')]
```

SUGGESTION

All the utilities could be put in a separate utils or initialization file.

```

# Diagonals Units

# Option 1: Using zip function
# diagonal_units1 = [a[0]+a[1] for a in zip(rows, cols)]
# diagonal_units2 = [a[0]+a[1] for a in zip(rows, cols[::-1])]

# Option 2: More easily understood
diagonal_units1 = [[rows[i]+cols[i] for i in range(9)]]
diagonal_units2 = [[rows[i]+cols[8-i] for i in range(9)]]

unitlist = row_units + column_units + square_units + diagonal_units1 + diagonal_units2
```

AWESOME

Good job (y) Additional constraints for diagonal sudoku implemented successfully :)
You could implement this using list comprehension and zip in the foll way:-
diagonal_units = [[r+c for r,c in zip(rows,cols)], [r+c for r,c in zip(rows,cols[::-1])]]
To see more tips and tricks you could go to : http://www.petercollingridge.co.uk/book/export/html/362

```
units = dict((s, [u for u in unitlist if s in u]) for s in boxes)
peers = dict((s, set(sum(units[s],[]))-set([s])) for s in boxes)

def grid_values(grid):
    """
    Convert grid string of a Sudoku puzzle into a {<box>: <value>}
    dictionary representation with '123456789' value for empty values

    Args:
        grid: Sudoku grid in string form, 81 characters long
    Returns:
        Sudoku grid in dictionary form:
        - keys: Box labels, e.g. 'A1'
```

AWESOME

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

```
        - values: Value in corresponding box, e.g. '8', or '123456789' if it is empty.
        - i.e.
            {
```

```
108                 'A1': '123456789'
109                 'A2': '123456789',
110                 'A3': '3',
111                 'A4': '123456789',
112                 'A5': '2',
113                 ...
114                 'I9': '123456789'
115                }
116     """
117     values = []
118     all_digits = '123456789'
119     for c in grid:
120         if c == '.':
121             values.append(all_digits)
122         elif c in all_digits:
123             values.append(c)
124     assert len(values) == 81
125     return dict(zip(boxes, values))
126 
127 def display(values):
128     """
129     Display the values as a 2-D grid.
130     Args:
131         values(dict): The sudoku in dictionary form
132     """
133 
134     width = 1+max(len(values[s]) for s in boxes)
135     line = '+'.join(['-'*(width*3)]*3)
136     for r in rows:
137         print(''.join(values[r+c].center(width)+('|' if c in '36' else '')
138                       for c in cols))
139         if r in 'CF': print(line)
140     return
141 
142 def eliminate(values):
143     """
144     Input: Puzzle in dictionary form.
145     Output: Iterate over all boxes in puzzle that only have one value assigned to them,
146     remove this value from every one of its peers, and return puzzle in dictionary form
147     """
148     update_dict = values
149     for k, v in update_dict.items():
150         if len(update_dict[k]) == 1:
151             peer_keys = peers[k]
152             digit = update_dict[k]
153             for pk in peer_keys:
154                 # update_dict[pk] = update_dict[pk].replace(digit,'')
155                 # PyGame Attempt
156                 values = assign_value(values, pk, values[pk].replace(digit,''))
157 
158     return values
159 
160 def only_choice(values):
161     """
162     Finalize all values that are the only choice for a unit.
163 
164     Go through all the units/squares, and whenever there is a unit with
165     a box that contains an unsolved value that only fits in that one box,
166     assign the value to this box.
167 
168     Input: Sudoku in dictionary form.
169     Output: Resulting Sudoku in dictionary form after filling in Only Choices.
170     """
171     for unit in unitlist:
172         for digit in '123456789':
173             dplaces = [box
174                        for box in unit
175                        if digit in values[box]]
176             if len(dplaces) == 1:
177                 # values[dplaces[0]] = digit
178                 # PyGame Attempt
179                 values = assign_value(values, dplaces[0], digit)
180     return values
181 
182 
183 def naked_twins(values):
184     """
185     Eliminate values using the naked twins strategy. Find all instances of naked twins by:
186     - Find all boxes with exactly two possibilities by iterating over all boxes in puzzle.
187     - Storing in a list of tuples all pairs of boxes that each contain the same twin possibilities (naked twin
188     - Iterate over all the pairs of naked twins to:
189         - Find peer boxes that they have in common between them based on calculating their intersection
190         - With the set of intersecting peers determined, iterate over the set of intersecting peers and
191         delete the naked twins values from each of those intersecting peers that contain more than two possibl
192 
193     Args:
194         values(dict): a dictionary of the form {'box_name': '123456789', ...}
```

```
    Returns:
        the values dictionary with the naked twins eliminated from peers.
    """
```

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.

```
    # before = list(values)
    # print("Before naked twin:", values)

    # Find all boxes containing exactly two possibilities
    possible_twins = [box
                      for box in values.keys()
                      if len(values[box]) == 2]
    # print("Possible naked twins: ", possible_twins)

    # Store in list of tuples all pairs of boxes that each contain the same twin possibilities (naked twins)
    naked_twins = []
    for box_twin1 in possible_twins:
        for box_twin2 in peers[box_twin1]:
            if values[box_twin1] == values[box_twin2]:
                naked_twins.append((box_twin1, box_twin2))
    # print("Naked twins: ", naked_twins)
```

SUGGESTION

You should always remember to erase all debugging code that is not necessary to understand or run the code after you are done.

```

    # Iterate over all the pairs of naked twins.
    #   - Find peer boxes that they have in common between them based on calculating their intersection
    #   - Iterate over the set of intersecting peers
    #   - Delete the naked twins values from each of those intersecting peers that contain over two possible v
```

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.

```
    for index in range(len(naked_twins)):
        box1, box2 = naked_twins[index][0], naked_twins[index][1]
        peers1, peers2 = peers[box1], peers[box2]
        peers_intersection = set(peers1).intersection(peers2)
        for peer_box in peers_intersection:
            if len(values[peer_box]) > 2:
                for digit in values[box1]:
                    # values[peer_box] = values[peer_box].replace(digit, '')
                    # PyGame Attempt
                    values = assign_value(values, peer_box, values[peer_box].replace(digit,''))
    # print("After naked twin: ", values)
    return values

def reduce_puzzle(values):
    """
    Constraint Propagation and Only Choice Techniques applied.
    Input: Unsolved Sudoku puzzle as dict
    Process: Apply repeatedly the eliminate() and only_choice() functions
    as constraints. Stop and return puzzle when solved. Exit loop by returning
    False when stuck at box with no available values. If Sudoku puzzle unchanged
    after iterating both eliminate() and only_choice() functions then return the Sudoku
    Output: Solution to Sudoku puzzle as dict
    """
    stalled = False
    while not stalled:
        # Check how many boxes have a determined value
        solved_values_before = len([box
                                    for box in values.keys()
                                    if len(values[box]) == 1])
        # Use the Eliminate Strategy
        values = eliminate(values)
        # Use the Naked Twins Strategy
        values = naked_twins(values)
        # Use the Only Choice Strategy
```

AWESOME

Great job calling naked_twins from reduce puzzle.

```
255         values = only_choice(values)
256         # Check how many boxes have a determined value, to compare
257         solved_values_after = len([box
258                                    for box in values.keys()
259                                    if len(values[box]) == 1])
260         # If no new values were added, stop the loop.
261         stalled = solved_values_before == solved_values_after
262         # Sanity check, return False if there is a box with zero available values:
263         if len([box
264                 for box in values.keys()
265                 if len(values[box]) == 0]):
266             return False
267     return values
268 
269 def search(values):
270     """
271     Use Depth-First Search (DFS) and Constraint Propagation,
272     create a search tree and solve the Sudoku.
273     """
274     # First, reduce the puzzle with using the reduce_puzzle function
275     values = reduce_puzzle(values)
276     if values is False:
277         return False # Failed
278     if all(len(values[s]) == 1 for s in boxes):
279         return values # Solved!
280 
281     # Choose one of the unfilled squares with the fewest possibilities
282     # and extract the n,s (i.e. 2 possibilities at 'A' would return: 2, 'A')
283     n,s = min((len(values[s]), s)
284               for s in boxes
285               if len(values[s]) > 1)
286 
287     # Now use recursion to solve each one of the resulting Sudokus,
288     # and if one returns a value (not False), return that answer!
289     # i.e. loop for 8 and 9 when possibilities for a box is 89
290     for value in values[s]:
291         new_sudoku = values.copy() # copy of latest Sudoku puzzle with updates from calling reduce_puzzle func
292         new_sudoku[s] = value # modify copy (new search tree branch) with attempt at trying reduced possibilit
293         attempt = search(new_sudoku) # recursion using modified copy with new tree branch attempt
294         if attempt: # if does not return False or None from modified copy it returns the values from the attem
295             return attempt
296 
297 def solve(grid):
298     """
299     Find the solution to a Sudoku grid.
300     Args:
301         grid(string): a string representing a sudoku grid.
302             Example: '2.............62....1....7...6..8...3...9...7...6..4...4....8....52.............3'
303     Returns:
304         The dictionary representation of the final sudoku grid. False if no solution exists.
305     """
306     # Get Sudoku grid representation with unsolved boxes populated with possible values
307     values = grid_values(grid)
308     # display(values)
309     # Call recursive function that performs Depth First Search using Constraint Propagation techniques
310     # of Elimination and Only Choice to solve harder Sudoku problems including those using diagonals as peers
311     values = search(values)
312 
313     # print("values 1: ", values)
314 
315     if not isinstance(values, bool):
316         return values
317 
318 if __name__ == '__main__':
319     diag_sudoku_grid = '2.............62....1....7...6..8...3...9...7...6..4...4....8....52.............3'
320     # naked_twins_grid1 = '84.632.....34798257..518.6...6.97..24.8256..12..84.6...8..65..3.54.2.7.8...784.96'
321     # naked_twins_grid2 = '1.4.9..68956.18.34..84.695151.....868..6...1264..8..97781923645495.6.823.6.854179'
322     grid = diag_sudoku_grid
323 
324     values = solve(grid)
325 
326     # print("values 2: ", values)
327 
328     if values:
329         display(solve(grid))
330 
331     try:
332         from visualize import visualize_assignments
333         visualize_assignments(assignments)
334 
335     except SystemExit:
336         print('SystemExit occurred')
337     except:
338         print('We could not visualize your board due to a pygame issue. Not a problem! It is not a requirement
339 
```

- visualize.py
- solution_test.py
- objects/__init__.py
- objects/SudokuSquare.py
- objects/GameResources.py
- README.md
- PySudoku.py

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Student FAQ