UDACITY

PROJECT

Creating an AI Agent to solve Sudoku

A part of the Artificial Intelligence Nanodegree Program

PROJECT REVIEW

CODE REVIEW 5

NOTES

▾ solution.py 5

```
# AIND-Sudoku
assignments = []

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.
    """

    # Don't waste memory appending actions that don't actually change any values
    if values[box] == value:
        return values

    values[box] = value
    if len(value) == 1:
        assignments.append(values.copy())
    return values

def naked_twins(values):
    """Eliminate values using the naked twins strategy.
    Args:
        values(dict): a dictionary of the form {'box_name': '123456789', ...}

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

    # Find all instances of naked twins
    # Eliminate the naked twins as possibilities for their peers
    twins_values = [box for box in values.keys() if len(values[box]) == 2]
    #print(twins_values)
    for box in twins_values:
        digit = values[box]
        # Find twins in row
        twins_row = [row for row in row_peers[box] if values[row] == digit]
        if len(twins_row) >= 1:
            for row in list(set(row_peers[box]) - set(twins_row)):
                for d in digit:
                    values[row] = values[row].replace(d,'')
                    #assign_value(values, peer, values[peer].replace(d,''))
        # Find twins in column
        twins_column = [column for column in column_peers[box] if values[column] == digit]
        if len(twins_column) >= 1:
            for column in list(set(column_peers[box]) - set(twins_column)):
                for d in digit:
                    values[column] = values[column].replace(d,'')
        # Find twins in square
        twins_square = [square for square in square_peers[box] if values[square] == digit]
        if len(twins_square) >= 1:
            for square in list(set(square_peers[box]) - set(twins_square)):
                for d in digit:
                    values[square] = values[square].replace(d,'')
        # Find twins in diagonal
        twins_diagonal_p = [diagonal for diagonal in diagonal_peers_p[box] if values[diagonal] == digit]
        if len(twins_diagonal_p) >= 1:
            for diagonal in list(set(diagonal_peers_p[box]) - set(twins_diagonal_p)):
                for d in digit:
```

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.

```
                    values[diagonal] = values[diagonal].replace(d,'')
        twins_diagonal_n = [diagonal for diagonal in diagonal_peers_n[box] if values[diagonal] == digit]
        if len(twins_diagonal_n) >= 1:
            for diagonal in list(set(diagonal_peers_n[box]) - set(twins_diagonal_n)):
                for d in digit:
                    values[diagonal] = values[diagonal].replace(d,'')
    return values

def cross(A, B):
    "Cross product of elements in A and elements in B."
    return [s+t for s in A for t in B]

boxes = cross(rows, cols)
row_units = [cross(r, cols) for r in rows]
```

SUGGESTION

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

```
column_units = [cross(rows, c) for c in cols]
square_units = [cross(rs, cs) for rs in ('ABC','DEF','GHI') for cs in ('123','456','789')]
diagonal_units_p = [['A1', 'B2', 'C3', 'D4', 'E5', 'F6', 'G7', 'H8', 'I9']]
```

AWESOME

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

```
diagonal_units_n = [['A9', 'B8', 'C7', 'D6', 'E5', 'F4', 'G3', 'H2', 'I1']]
```

SUGGESTION

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

```
unitlist = row_units + column_units + square_units + diagonal_units_p + diagonal_units_n

row_dic = dict((s, [u for u in row_units if s in u]) for s in boxes)
col_dic = dict((s, [u for u in column_units if s in u]) for s in boxes)
squares_dic = dict((s, [u for u in square_units if s in u]) for s in boxes)
diagonal_dic_p = dict((s, [u for u in diagonal_units_p if s in u]) for s in boxes)
diagonal_dic_n = dict((s, [u for u in diagonal_units_n if s in u]) for s in boxes)
units = dict((s, [u for u in unitlist if s in u]) for s in boxes)

row_peers = dict((s, set(sum(row_dic[s],[]))-set([s])) for s in boxes)
column_peers = dict((s, set(sum(col_dic[s],[]))-set([s])) for s in boxes)
square_peers = dict((s, set(sum(squares_dic[s],[]))-set([s])) for s in boxes)
diagonal_peers_p = dict((s, set(sum(diagonal_dic_p[s],[]))-set([s])) for s in boxes)
diagonal_peers_n = dict((s, set(sum(diagonal_dic_n[s],[]))-set([s])) for s in boxes)
peers = dict((s, set(sum(units[s],[]))-set([s])) for s in boxes)
#print(len(peers['A1']))

def grid_values(grid):
    """
    Convert grid into a dict of {square: char} with '123456789' for empties.
    Args:
        grid(string) - A grid in string form.
    Returns:
        A grid in dictionary form
            Keys: The boxes, e.g., 'A1'
            Values: The value in each box, e.g., '8'. If the box has no value, then the value will be '123456789'.
    """
    chars = []
    digits = '123456789'
    for c in grid:
        if c in digits:
            chars.append(c)
        if c == '.':
            chars.append(digits)
    assert len(chars) == 81
    return dict(zip(boxes, chars))

def display(values):
    """
    Display the values as a 2-D grid.
    Args:
        values(dict): The sudoku in dictionary form
    """
    #print (values)
    if values == False or None:
        print ("False")
        return

    width = 1+max(len(values[s]) for s in boxes)
    line = '+'.join(['-'*(width*3)]*3)
    for r in rows:
        print(''.join(values[r+c].center(width)+('|' if c in '36' else '')
                      for c in cols))
        if r in 'CF': print(line)
    return

def eliminate(values):
    """
    Go through all the boxes, and whenever there is a box with a value, eliminate this value from the values of all its peers.
    Input: A sudoku in dictionary form.
    Output: The resulting sudoku in dictionary form.
    """
    solved_values = [box for box in values.keys() if len(values[box]) == 1]
    for box in solved_values:
        digit = values[box]
        for peer in peers[box]:
            values[peer] = values[peer].replace(digit,'')
            #assign_value(values, peer, values[peer].replace(digit,''))
    return values

def only_choice(values):
    """
    Go through all the units, and whenever there is a unit with a value that only fits in one box, assign the value to this box.
    Input: A sudoku in dictionary form.
    Output: The resulting sudoku in dictionary form.
    """
    for unit in unitlist:
        for digit in '123456789':
            dplaces = [box for box in unit if digit in values[box]]
            if len(dplaces) == 1:
                values[dplaces[0]] = digit
                #assign_value(values, dplaces[0], digit)
    return values

def reduce_puzzle(values):
    """
    Iterate eliminate() and only_choice(). If at some point, there is a box with no available values, return False.
    If the sudoku is solved, return the sudoku.
    If after an iteration of both functions, the sudoku remains the same, return the sudoku.
    Input: A sudoku in dictionary form.
    Output: The resulting sudoku in dictionary form.
    """
    solved_values = [box for box in values.keys() if len(values[box]) == 1]
    stalled = False
    while not stalled:
        solved_values_before = len([box for box in values.keys() if len(values[box]) == 1])
        values = eliminate(values)
        values = only_choice(values)
        #display(values)
        #print('\n\n')
        values = naked_twins(values)
```

AWESOME

Good job calling the naked_twins here, as it is 3rd strategy to reduce the search space.

```
        #display(values)
        #values = only_choice(values)
        solved_values_after = len([box for box in values.keys() if len(values[box]) == 1])
        stalled = solved_values_before == solved_values_after
        if len([box for box in values.keys() if len(values[box]) == 0]):
            return False
    return values

def search(values):
    "Using depth-first search and propagation, try all possible values."
    values = reduce_puzzle(values)
    #return(values)

    if values is False:
        return False ## Failed earlier
    if all(len(values[s]) == 1 for s in boxes):
        return values ## Solved!
    # Choose one of the unfilled squares with the fewest possibilities
    n,s = min((len(values[s]), s) for s in boxes if len(values[s]) > 1)
    # Now use recurrence to solve each one of the resulting sudokus, and
    for value in values[s]:
        new_sudoku = values.copy()
        new_sudoku[s] = value
        #assign_value(new_sudoku, s, value)
        attempt = search(new_sudoku)
        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)
    #display(values)
    return search(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 = '.....8..1..1............5.......3...6.3..52....2...3.3..4...6.51...9.......'
    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

Rate this review

Student FAQ