UDACITY

PROJECT

Creating an AI Agent to solve Sudoku

A part of the Artificial Intelligence Nanodegree and Specializations Program

PROJECT REVIEW

CODE REVIEW 4

NOTES

▸ README.md 2

▾ solution.py 2

```python
assignments = []

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
    for unit in unitlist:
        reversed_shorted_values = dict()
        for box in unit:
            if len(values[box])==2:
                if values[box] in reversed_shorted_values.keys():
                    reversed_shorted_values[values[box]].append(box)
                else:
                    reversed_shorted_values[values[box]] = [box]
        for two_digit in reversed_shorted_values.keys():
            if len(reversed_shorted_values[two_digit]) > 1:
                #only the first two found are twins, existence or more indicate puzzle will go to false case very soon
                keep0 = reversed_shorted_values[two_digit][0]
                keep1 = reversed_shorted_values[two_digit][1]
                for box in unit:
                    if box != keep0 and box != keep1:
                        assign_value(values, box, values[box].replace(two_digit[0],''))
                        assign_value(values, box, values[box].replace(two_digit[1],''))
    return values

rows, cols= 'ABCDEFGHI', '123456789'
size = 9
def cross(A, B):
    "Cross product of elements in A and elements in B."
    return [a+b for a in A for b in B]

boxes = cross(rows, cols)
diagonal_units = [[rows[i]+cols[i] for i in range(size)],[rows[i]+cols[-i-1] for i in range(size)]]
row_units = [cross(rows[i], cols) for i in range(size)]
column_units = [cross(rows, cols[i]) for i in range(size)]
square_units = [cross(rs, cs) for rs in ('ABC','DEF','GHI') for cs in ('123','456','789')]
unitlist = diagonal_units+row_units + column_units + square_units
units = dict((box, [unit for unit in unitlist if box in unit]) for box in boxes)
peers = dict((box, set(sum(units[box],[]))-set([box])) for box in boxes)
```

```
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'.
    """
    grid_choice = ['123456789' if value == '.' else value for value in grid]
    return dict(zip(boxes, grid_choice))

def display(values):
    """
    Display the values as a 2-D grid.
    Args:
        values(dict): The sudoku in dictionary form
    """
    width = 1 + max(len(values[box]) for box in boxes)
    line = '+'.join(['-'*(width*3)]*3)
    for row in rows:
        print(''.join(values[row+col].center(width)+('|' if col in '36' else '')
                      for col in cols))
        if row in 'CF': print(line)
    return

def eliminate(values):
```

REQUIRED

Provide your method with a docstring that helps in understanding the functioning of the method, provide a docstring to every method you define in the code.

```
    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]:
            assign_value(values, peer, values[peer].replace(digit,''))
    return values

def only_choice(values):
    for unit in unitlist:
        for digit in '123456789':
            hits = [box for box in unit if digit in values[box]]
            if len(hits) == 1:
                assign_value(values, hits[0], digit)
    return values

def reduce_puzzle(values):
    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)
```

REQUIRED

Add some inline comments throughout the code.

```
        values = naked_twins(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, create a search tree and solve the sudoku."
    # First, reduce the puzzle using the previous function
    values = reduce_puzzle(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
    tolerance, box = min((len(values[s]), s) for s in boxes if len(values[s]) > 1)
    # Now use recursion to solve each one of the resulting sudokus, and if one returns a value (not False), return that answer!
    for i in range(tolerance):
        new_values = values.copy()
        new_values[box] = values[box][i]
        attempt = search(new_values)
        if attempt:
            return attempt
    return False
    # If you're stuck, see the solution.py tab!

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.
    """
    return search(grid_values(grid))

if __name__ == '__main__':
    diag_sudoku_grid = '2.............62....1....7...6..8...3...9...7...6..4...4....8....52.............3'
    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.')

```

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Student FAQ