AI - 7 Nov. 2018

Judicious brute force

There is a class of puzzles where each position of the puzzle is to be filled in with a symbol. This could be as simple as a 0/1 symbol (such as in the n-queens problem where a 0 means no queen at that position and a 1 means there is a queen at that position), or numbers (such as 1-9 in most Sudoku problems), or arbitrary labels (such as 0-9 and A-F in 16 x 16 sudokus). Often, as with a standard Sudoku puzzle, the numbers are acting as only labels, meaning that the numeric qualities of the number are not being used. In those cases, letters or other symbols could just as easily be used.

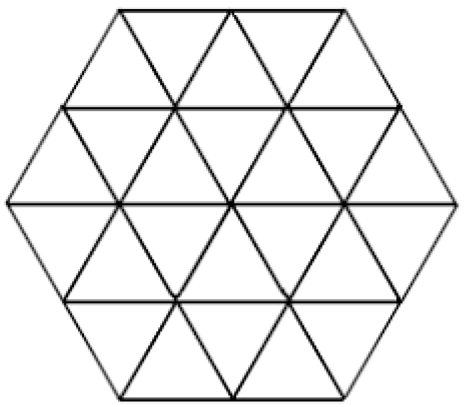
A possible (horrible) approach to solving this type of puzzle might be

```
def bruteForce(pzl):
  # returns a solved pzl or the empty string on failure
  if pzl is completely filled out:
    return "" if isInvalid(pzl) else pzl

  find a setOfChoices that is collectively exhaustive
  for each possibleChoice in the setOfChoices:
    subPzl = pzl with possibleChoice applied
    bF = bruteForce(subPzl)
    if bF: return subPzl
  return ""
```

The reason this is so horrible is that it fills each puzzle out completely before testing. For all but the most trivial of problems, the routine will hang because there are way too many possibilities to recurse through. However, we can gain considerable traction with only a slight modification. By testing the validity of a potential solution at each step of the process, it is the hope that large subtrees of invalid possibilities may be immediately pruned. For example, in the n-queens problem, if two queens attacked each other, there is no reason to continue filling in the puzzle since it's already unsolvable. The following is what was shown in class as our starting out point for the month:

```
def bruteForce(pzl):
  # returns a solved pzl or the empty string on failure
  if isInvalid(pzl): return ""
  if isSolved(pzl):  return pzl

  find a setOfChoices that is collectively exhaustive
  for each possibleChoice in the setOfChoices:
    subPzl = pzl with possibleChoice applied
    bF = bruteForce(subPzl)
    if bF: return subPzl
  return ""
```

Your homework is to use the above `bruteForce` code (or small variation on it) to solve the following questions. The expectation is to put in an honest effort (meaning no more than 2 hours):



You are given an inner hexagon (made up of 6 small triangles), and around each of the 6 outer vertices of the hexagon you construct another hexagon to wind up with a total of 7 complete hexagons utilizing 24 of the small triangles in total. In other words, the outer perimeter has 18 triangles.

In addition, for each of the three main directions for this amalgamation of triangles, there are 4 rows of either 5 or 7 triangles. That is, there are 12 distinct rows of triangles.

Q1) Is there a way to label each small triangle with an integer from 1-6 where none of the seven hexagons has two triangles with the same number?

Q2) Is there a way to label each small triangle with an integer from 1-7 where none of the seven hexagons and none of the twelve rows of triangles has two triangles with the same number?

If the answer to either question is yes, you should provide a solution.

Considerations:
Your program is not expected to draw triangles or hexagons. However, it must have some way of identifying each of the triangles. So, you will have to devise some indexing scheme to decide which is the 0 triangle, which is triangle with index 1, and so forth. As long as each triangle has a unique index, it would be a valid indexing (you could even give each triangle a name, like "Fred" and "Bill" and "Martha", but it's not clear to me what purpose this would serve).

What is important is what sets of triangles constitute a hexagon, or a row of triangles. If you make a dictionary of each such set, then your code should run just fine. Besides representing your puzzle as a string or list, it's the only data structure you'll need.

Use a period (.) to represent an unfilled position in the puzzle. This is because _ and spaces tend to merge when displayed while the periods will stay distinct.

In your code, I expect to see an explicit `isInvalid(pzl)` and `bruteForce(pzl)` function. What you should **turn in** to me at the **start of class** on **Friday** is a printout of your code and the answers to the two questions, if you are able to find them (if not, then just let me know what issue your code is having).

If your answer to either of Q1 or Q2 is yes, then draw an image of the triangle amalgamation and fill it in with the solution. I don't need to see your indexing scheme since it's internal to your program, but it is also OK to write the index in tiny script in the triangles if you wish. In this latter case, make sure that your indices are not competing with the labels you've found).

Note: There is no late work accepted on this assignment. If you are done in class, you should apply your technique to solving the wood puzzle shown at the start of end of the quarter.