3D Squares

Jesse Friedman



Date: 3/04/18

**Table of Contents:**

**Initial Observations**……………………………………………………………………………………………..2

**Approach/pseudo code**……………………………………………………………………………..………..6

**Problems with approach**………………………………………………………………………………………6

**Code**……………………………………………………………………………………………………………………..7

**Sample output.**…………………………………………………………………………………………………….10

**Conclusion**…….……………………………………………………………………………………………………..12

**Updated conclusion**……………………………………………………………………………………..………12

1. **Initial Observations:**
2. There are 9 square puzzle pieces organized on a board in a 3 x 3 configuration (i.e. 3 columns, 3 rows).
3. Each piece can be placed in any of the 9 locations on the board and rotated in any of 4 directions (i.e. N, E, S, and W). I label these 4 directions as 0 – 3, as illustrated below:

0

3.

1

2.

1. Each piece has either an image of an insect head or insect tail on all 4 sides of the piece, totaling 36 insect body parts.
2. There are 4 insects most easily identifiable by their colors. These include:

Purple, green/orange, red/black, yellow/orange

1. In order for a match to occur, 2 pieces must align such that a particular insect has a matching head and tail. There is a maximum of 12 matching insects.
   1. There isn’t an even distribution of body parts within each insect, except for the purple insect (Chart A). For example, the green/orange insect has 3 heads and 6 tails. As a result, only 3 total green/orange bugs can exist.
   2. All insects do not have the same number of heads plus tails (Chart A). For example, the purple insect has 10 total body parts whereas the yellow/orange insect has 8.

Chart A

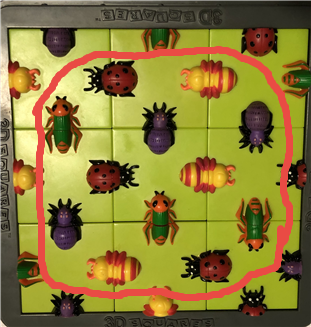
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Purple insect** | **Green/orange** | **Red/black** | **Yellow/orange** | **Total** |
| **Heads** | 5 | 3 | 5 | 5 | 18 |
| **Tails** | 5 | 6 | 4 | 3 | 18 |
| **Total** | 10 | 9 | 9 | 8 | 36 |

1. After assigning each insect body part a unique value (1 - 8), it becomes evident that the value of each insect’s head or tail has a difference of 4 (Chart B).

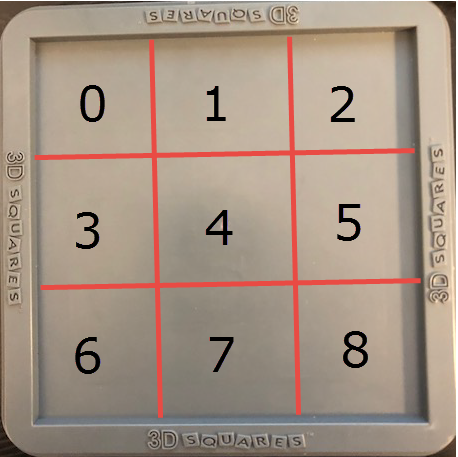
Chart B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Purple insect** | **Green/orange** | **Red/black** | **Yellow/orange** |
| **Heads** | 1 | 2 | 3 | 4 |
| **Tails** | 5 | 6 | 7 | 8 |
| **Difference** | 4 | 4 | 4 | 4 |

1. Nothing can match with body parts that are on the perimeter of the board. The 12 maximum matches all exist within the interior of the board, as illustrated below.



1. Each space (not piece) on the board is a assigned a value, as follows:



1. With the locations on the board labeled 0 – 8 and the direction that a piece can be rotated labeled 0 – 4, the follow observation(s) can be made:
   1. If a match is trying to be made with a body part in the 0 direction (see #2) and the piece is in location 0 – 8, the possibilities are as follows:

|  |  |  |
| --- | --- | --- |
| **Piece in location #** | **Can be matched with piece in location #** | **Direction of body part in matched piece** |
| 0 | Not possible (on perimeter) | -- |
| 1 | Not possible (on perimeter) | -- |
| 2 | Not possible (on perimeter) | -- |
| 3 | 0 | In 2nd direction |
| 4 | 1 | In 2nd direction |
| 5 | 2 | In 2nd direction |
| 6 | 3 | In 2nd direction |
| 7 | 4 | In 2nd direction |
| 8 | 5 | In 2nd direction |

* 1. If a match is trying to be made with a body part in the 1 direction (see #2) and the piece is in location 0 – 8, the possibilities are as follows:

|  |  |  |
| --- | --- | --- |
| **Piece in location #** | **Can be matched with piece in location #** | **Direction of body part in matched piece** |
| 0 | 1 | In 3rd direction |
| 1 | 2 | In 3rd direction |
| 2 | Not possible (on perimeter) | -- |
| 3 | 4 | In 3rd direction |
| 4 | 5 | In 3rd direction |
| 5 | Not possible (on perimeter) | -- |
| 6 | 7 | In 3rd direction |
| 7 | 8 | In 3rd direction |
| 8 | Not possible (on perimeter) | -- |

* 1. If a match is trying to be made with a body part in the 2 direction (see #2) and the piece is in location 0 – 8, the possibilities are as follows:

|  |  |  |
| --- | --- | --- |
| **Piece in location #** | **Can be matched with piece in location #** | **Direction of body part in matched piece** |
| 0 | 3 | In 0th direction |
| 1 | 4 | In 0th direction |
| 2 | 5 | In 0th direction |
| 3 | 6 | In 0th direction |
| 4 | 7 | In 0th direction |
| 5 | 8 | In 0th direction |
| 6 | Not possible (on perimeter) | -- |
| 7 | Not possible (on perimeter) | -- |
| 8 | Not possible (on perimeter) | -- |

* 1. If a match is trying to be made with a body part in the 3 direction (see #2) and the piece is in location 0 – 8, the possibilities are as follows:

|  |  |  |
| --- | --- | --- |
| **Piece in location #** | **Can be matched with piece in location #** | **Direction of body part in matched piece** |
| 0 | Not possible (on perimeter) | -- |
| 1 | 0 | In 1st direction |
| 2 | 1 | In 1st direction |
| 3 | Not possible (on perimeter) | -- |
| 4 | 3 | In 1st direction |
| 5 | 4 | In 1st direction |
| 6 | Not possible (on perimeter) | -- |
| 7 | 6 | In 1st direction |
| 8 | 7 | In 1st direction |

1. **Approach/pseudo code**

We now have the basis to start developing rules for our algorithm:

1. Examine each in a position (0-8) on the board
   1. Start with the body part in position (direction) 0 on the piece
      1. Compare this to the other piece and the position of the body part on this other piece, that corresponds to the piece and position that’s being examined (see #10 under Observations).
         1. If the difference between the two body parts under observation have a difference in their values of 4, we have a match:
            1. Increment a match counter.
         2. If the difference is not 4, we don’t have a match:
         3. Break loop because the configuration is not perfect (there aren’t 12 matches of heads and tails). Log the configuration of pieces and the match counter.
2. Continue to do this for every possible combination of a direction of a given piece and the location of that piece on the board.
3. **Problems with this approach**

There are 4 different ways that each piece can be rotated and each piece with a given rotation can be place in 9 different locations on the board. This results in 95,126,814,720 different combinations of location and rotation!

This is definitely a CPU intensive operation and not one for my aging, 4 core desktop. The only way to get through this many operations, in a timely manner, is with a more powerful computer (via additional cores). This can be further improved with multiple computers working simultaneously.

As a result, I initiated 2 Google cloud servers, with 8 cores each, to run through the combinations. I had one start from the beginning of the potential combinations and one at the end, such that they aren’t working on the same combinations. Theoretically, the number of combinations can be dissected even further and subsequently ran on additional computers, thereby reducing the overall runtime.

I also implemented multi-processing in Python to maximize CPU allocation to the program. Accordingly, depending on the type of computer that the script is run on, the number of processes may need to be adjusted.

1. **Code (also on GitLab)**

**import** itertools

**from** multiprocessing **import** Pool

#####################################################################

# def piece\_match(position\_num)

# Purpose: as an input, it takes the position number on the piece under

# observation as an input and returns the other pieces and location of

# the body part that the position number of the input corresponds to

# Pre-conditions: position\_num is between 0 and 3

# Post-conditions: none

######################################################################

**def** piece\_match(position\_num):

**if** position\_num == 0:

        match\_list = [[3, 0], [4, 1], [5, 2], [6, 3], [7, 4], [8, 5]]  # if looking for a match for position 0 (top), piece 3 & 0 must go together, etc.

        pos\_of\_matching\_piece = 2                                      # with a body part in position 0, the matching body part is in position 2 on the other piece

**elif** position\_num == 1:

        match\_list = [[0, 1], [1, 2], [3, 4], [4, 5], [6, 7], [7, 8]]

        pos\_of\_matching\_piece = 3

**elif** position\_num == 2:

        match\_list = [[0, 3], [1, 4], [2, 5], [3, 6], [4, 7], [5, 8]]

        pos\_of\_matching\_piece = 0

**elif** position\_num == 3:

        match\_list = [[1, 0], [2, 1], [4, 3], [5, 4], [7, 6], [8, 7]]

        pos\_of\_matching\_piece = 1

**return** match\_list, pos\_of\_matching\_piece

#######################################################################

# def spin\_pieces(puzzle\_pieces)

# Purpose: as an input, it takes a shuffled puzzle\_pieces\_initial

# (see main) and iterates through all of the combinations of this list

# sending one iteration at a time to match\_count

# Pre-conditions: puzzle\_pieces is a list that has 9 sub-lists

# representing each piece and its 4 body part IDs (1 - 8).

# Post-conditions: none

#######################################################################

**def** spin\_pieces(puzzle\_pieces):

    spin\_combos = list(itertools.product([0, 1, 2, 3], repeat=9)) # produces a template of all of the combinations of 4 rotations on 9 different pieces

    perm\_list = []

**for** num\_combos **in** range(len(spin\_combos)):                    # produces a list of 4 rotations on different pieces using the template produced by spin\_combos

        counter = 0

        perm\_list\_temp = []

**for** j **in** spin\_combos[num\_combos]:

            perm\_list\_temp.append(puzzle\_pieces[counter][j])

**if** counter == 8:

**break**

            counter += 1

            perm\_list.append(perm\_list\_temp)

    match\_count(perm\_list) # call match\_count with the created

#####################################################################

# def match\_count(perm\_list)

# Purpose: takes a given configuration of pieces on the board as an input

# and logs the number of matches

# Pre-conditions: perm\_list is a list of all of the combinations of

# rotations for a given layout

# Post-conditions: none

######################################################################

**def** match\_count(perm\_list):

    match\_counter=0

    counter=0

**for** sets **in** range (len(perm\_list)):

        counter+=1

**if** match\_counter >= 18:                                                  # counts each match twice. So this is logging each configuration with at least 9 matches on the board

            text\_file = open("results.txt", "w")                                 # write the number of matches and the configuration of pieces (the layout) to a file

            text\_file.write(str(match\_counter) + " " + str(perm\_list[sets]))

            text\_file.close()

**print**('Match counter:', match\_counter)                               # also output it to command line

**print**(perm\_list[sets])

        match\_counter = 0

**for** side\_of\_piece **in** range(4):

            match\_list, pos\_of\_matching\_piece = piece\_match(side\_of\_piece)

**for** j **in** range(len(match\_list)):

                value\_in\_piece\_A = (perm\_list[sets][match\_list[j][0]][side\_of\_piece])             # assigns the ID of the body part of the piece under observation

                value\_in\_piece\_B = (perm\_list[sets][match\_list[j][1]][pos\_of\_matching\_piece])     # assigns the ID of the body part of the piece being compared to, based on its location

**if** abs(value\_in\_piece\_A - value\_in\_piece\_B) != 4:                                 # if the difference isn't equal to 4, break the loop and move into next board layout

**break**

**if** abs(value\_in\_piece\_A-value\_in\_piece\_B) == 4:                                   # if the difference is 4, the pieces match and increment match\_counter

                    match\_counter=match\_counter+1

#####################################################################

# MAIN

#####################################################################

location\_combos = list(itertools.permutations([0,1,2,3,4,5,6,7,8]))             # produces a template of all of the permutations of 9 different values (0 - 8)

# sets an initial layout of the pieces on the board. Each piece has 4 values representing the ID of the body part in a particular direction

# here piece 0 has an initial value of 5, 8, 2, 3. Thus it has a purple tail in position 0 (top), yellow/orange tail in position 1 (right), etc. It is than "rotated 3 times".

puzzle\_pieces\_initial =  [

[ [5,8,2,3], [3,5,8,2], [2,3,5,8], [8,2,3,5] ],#0

                 [ [2,1,3,7], [7,2,1,3], [3,7,2,1], [1,3,7,2] ],#1

                 [ [8,7,6,5], [5,8,7,6], [6,5,8,7], [7,6,5,8] ],#2

                 [ [6,7,1,4], [4,6,7,1], [1,4,6,7], [7,1,4,6] ],#3

                 [ [6,4,1,7], [7,6,4,1], [1,7,6,4], [4,1,7,6] ],#4

                 [ [5,3,2,4], [4,5,3,2], [2,4,5,3], [3,2,4,5] ],#5

                 [ [6,3,4,8], [8,6,3,4], [4,8,6,3], [3,4,8,6] ],#6

                 [ [1,6,5,4], [4,1,6,5], [5,4,1,6], [6,5,4,1] ],#7

                 [ [1,3,6,4], [4,1,3,6], [6,4,1,3], [3,6,4,1] ],#8

                 ]

perm\_list\_temp=[]

perm\_list=[]

**for** a **in** range(len(location\_combos)):                                           # produces a list of all of the combinations of layouts with a given piece's rotational configuration AND placement on the board. So each element in the list is puzzle\_pieces\_initial, shuffled.

    counter=0

    perm\_list\_temp = []

**for** j **in** location\_combos[a]:

        perm\_list\_temp.append(puzzle\_pieces\_initial[j])

        perm\_list.append(perm\_list\_temp)

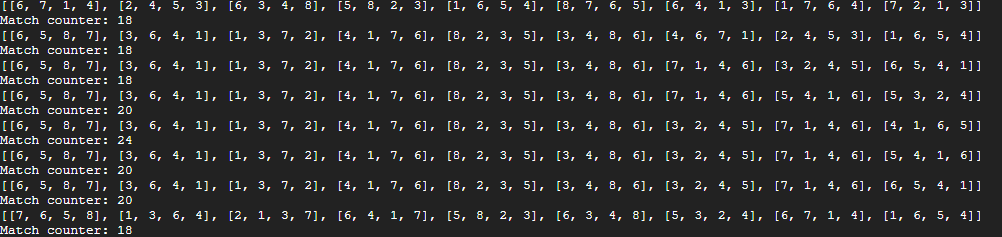
**if** \_\_name\_\_ == '\_\_main\_\_':

    p = Pool(processes=15)                                                      # use multiprocessing to maximize CPU allocation and reduce overall runtime (hopefully without crashing computer). 15 may be too high/low depending on the computer.

    data = p.map(spin\_pieces, [perm\_list[i] **for** i **in** range(len(perm\_list))]  )

    p.close()

1. **Sample output:**



Translation of output:

1. “Match counter” is obviously the number of matches that were made by the configuration of pieces listed above it. Each match is effectively counted twice, thus the real number of matches is the number seen divided by 2 (this can adjusted in the code). So 18 matches is really 9 matches and 24 matches is a perfect board (i.e. 12 matches).
2. Each piece is represented by the 4 numbers inside the single brackets (“[ ]”). There are a series of 9 of these in a given row representing the 9 pieces in the puzzle and the order in which they appear on the board (see #9 in Initial Observations).
   1. Each of the 4 numbers in a bracket represent a body part ID (see #7, chart B in Initial Observations).
   2. The order of the 4 numbers represent the direction of the body part on the piece (see #2 in Initial Observations).

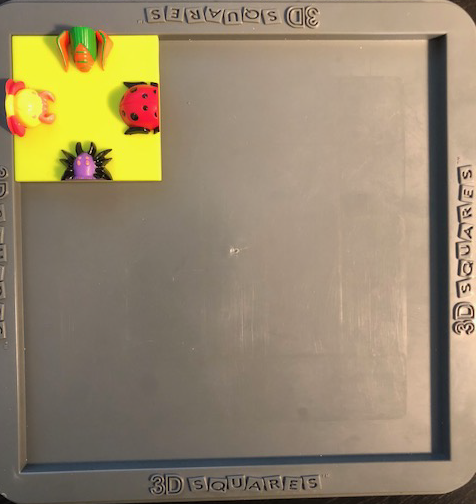
Example:



[6, 7, 1, 4] is the first piece and therefore in position 0 on the board. It consists of:

* a green/orange tail in direction 0
* a red/black tail in direction 1
* a purple head in direction 2
* a yellow/orange head in direction 3

This is the equivalent of the following on the board:



1. **Conclusion**

The approach that I took to solve this puzzle algorithmically was rather brute force like, where every combination of piece placement and rotation was analyzed. This is sure to find the solution(s) to the puzzle, but given the sheer number of combinations, probably isn’t the most efficient (time and/or CPU wise).

While I’m confident that a superior approach exists, I’m not sure that it can be conceived without first determining the configurations of pieces that lead to a “perfect board” (i.e. 12 matches). Now that I have several configurations that lead to a “perfect board”, I can see what they have in common (albeit a small sample size) and extrapolate some ideas to increase the efficiency of the algorithm. Perhaps certain pieces are more important (e.g. #6 in Initial Observations) insofar as where they must be positioned on the board and in what direction.

As of this writing, there are still combinations remaining to be processed, however the summary of matches thus far is as follows:

|  |  |
| --- | --- |
| **Match count** | **Number of combinations** |
| 9 | 3498 |
| 10 | 360 |
| 11 | 25 |
| 12 (“perfect board”) | 4\* (See G) |

1. **Updated conclusion:**

Instead of revising the conclusion above, I thought that simply updating it would better reflect the evolution of my thought process on this. After carefully observing the “perfect boards” that the algorithm found, there is arguably only 1 solution. The 4 solutions that it found have the same layout of pieces with identical directions but each is a representation of the entire board turned 90 degrees. **Absolutely fascinating**. In other words:

1. Perfect board A:

[[3, 7, 2, 1], [4, 8, 6, 3], [1, 6, 5, 4], [6, 4, 1, 3], [2, 3, 5, 8], [1, 4, 6, 7], [5, 8, 7, 6], [1, 7, 6, 4], [2, 4, 5, 3]]



1. Perfect board B = Perfect board A, rotated 90 degrees clockwise (can this be considered another solution? The algorithm thinks so):

[[6, 5, 8, 7], [3, 6, 4, 1], [1, 3, 7, 2], [4, 1, 7, 6], [8, 2, 3, 5], [3, 4, 8, 6], [3, 2, 4, 5], [7, 1, 4, 6], [5, 4, 1, 6]]



1. Perfect board C = Perfect board A, rotated 180 degrees (can this be considered another solution? The algorithm thinks so):

[[6, 5, 8, 7], [3, 6, 4, 1], [1, 3, 7, 2], [4, 1, 7, 6], [8, 2, 3, 5], [3, 4, 8, 6], [3, 2, 4, 5], [7, 1, 4, 6], [5, 4, 1, 6]]



1. Perfect board D = Perfect board A, rotated 270 degrees clockwise (can this be considered another solution? The algorithm thinks so):

[[6, 5, 8, 7], [3, 6, 4, 1], [1, 3, 7, 2], [4, 1, 7, 6], [8, 2, 3, 5], [3, 4, 8, 6], [3, 2, 4, 5], [7, 1, 4, 6], [5, 4, 1, 6]]

