CS 7637

Project 2 Reflection

Daniel Rozen

drozen3

GTid: 903104846

Project 2 Reflection

(questions being answered are in italics)

- *How does your agent reason over the problems it receives? What is its overall problem-solving process?*

  My agent reasons over the problems it receives by first representing all of the objects in the matrix and answers verbally in a Semantic Network. It then uses the Generate and Test method in order to loop through each potential answer (figures 1 to 6), placing the answer in figure D for 2x2 matrices or I for 3x3 matrices and then measures the relationships and transformations from objects in 3 other corresponding figures, such as for 2x2: A to B and figures C to D or for 3x3: from figures A to C and from figures G to I. It then compares these transformations to each other, eg. comparing (A to B) to (C to D). My agent computes vertical, horizontal, and diagonal transformation comparisons. Since we often have multiple potentially close answers the agent then uses similarity weights in order to choose the best match. These were chosen based on importance of similarity as will be explained in the next section on selecting an answer.

- *Did you take any risks in the design of your agent, and did those risks pay off?*

  The risks I took in designing my agent involve that I first attempted to extend my Project 1 agent from answering 2x2 matrices to answering 3x3 matrices by simply solving them as 2x2 problems looking only at A, C, and G as was suggested by Dr. David Joyner. This made sense as I often noticed the transformation from A to B was similar to B to C, and also A to D was similar to D to G (ie. they seemed to be linear). However, this risk didn't initially pay off, as my agent

performed poorly on the 3x3 C problems, only correctly answering 3/12 of the C basic problems (3,4 and 9).

I then went on to solve each problem one by one as will be detailed below. This was in contrary to perhaps a better method of first creating a high level design that incorporates all of the problems and then bringing the design down to the code.

This risk eventually somewhat paid off in that I was able to successfully code an agent that solves 12/12 of the basic B problems and between 8-9/12 basic C problems.

- *How does your agent actually select an answer to a given problem? What metrics, if any, does it use to evaluate potential answers? Does it select only the exact correct answer, or does it rate answers on a more continuous scale?*

My agent uses metrics to assign a score to each of the potential answers, rating them on a continuous scale and then picks the answer with the highest score as the best answer. It does this by rating each answer, and subsequently checking if this is the highest rated answer yet, and if so, selects it as the new best answer. If there is a tie, the tie is broken with a random number generator of 0 or 1. 0 means take the new answer as the best answer.

Transformations were recorded, stored, and retrieved from a dictionary.

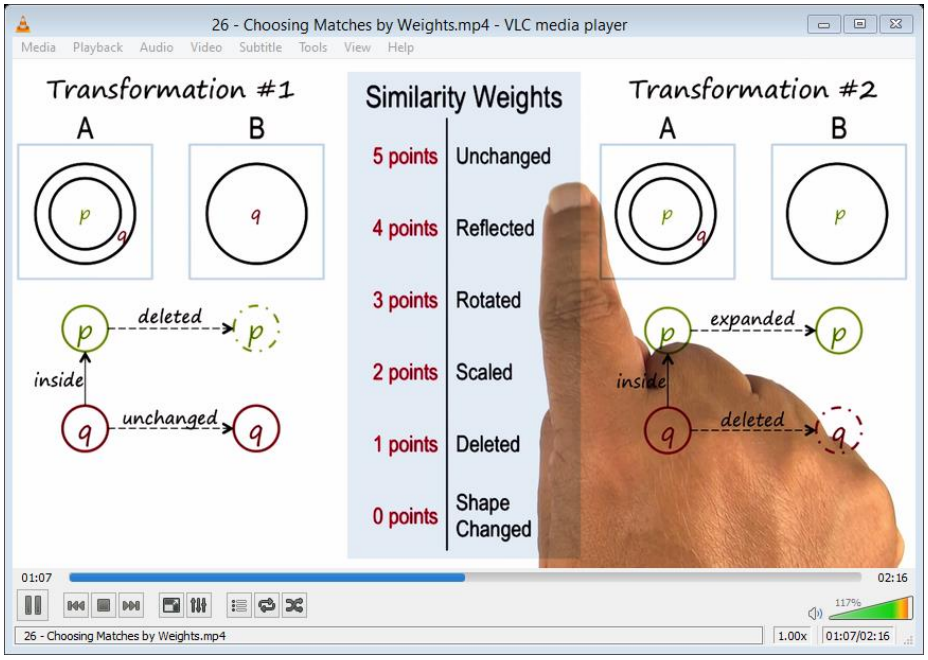
Correct transformations are assigned different points according to the transformation type in order to better be able to choose the best answer.

Here are the transformation weights that I chose:

```
# Define increment variables (Adjustable)
sameIncr = .1  # same general property increment value
angleIncr = 20   # same angle rotation transform increment value
reflectIncr = 15  # same reflection transform increment value
fillIncr = 40  # increment for correct fill transformation
shapeIncr = 100 # increment for correct shape transformation
deleteIncr = 200 # increment for correct deletion of objects
deleteCompIncr = 5 # increment for compensation for deleted/added objects having less
matches
alignIncr = 35 # increment for correct alignment transformation
sizeIncr = 50 # increment for correct size transformation
overlapsIncr = 21 # increment for correct overlap transformation
overlapsBoolIncr = 10 # increment for correct overlap boolean transformation
leftOfIncr = 20 # increment for correct left-of transformation
aboveIncr = 20 # increment for correct above transformation
```

These were chosen based on the importance of similar transformations as I perceived them in terms of difficulty of transformation as inspired by Lecture 3-26 (see figure below). More difficult transforms were given higher points if they matched, as the likelihood of them both occurring randomly was lower.

The weights were ranked in the following order of importance: deletion, shape, size, fill, alignment, rotation, reflection, overlaps, overlaps Boolean, and same properties.



The following transformations were discussed in my Project 1 reflection: similarity, reflection, rotation, shape, and deletion.

I would now like to discuss my agent's additions and improvements from Project 1:
My AI agent from project 1 required many coding improvements. As a beginner Python programmer, I spent much time refactoring my code in order to make it re-usable for comparison between different objects and also be extended to 3x3 problems. This involved extracting methods, creating new methods, tweaking the methods, and much time on debugging.

One of the major improvements that I spent much time on was Object Matching. This is something that is easily done by the human after a quick study of the figures. However it took me much time to figure out how to code object matching capabilities into my AI agent. It required many upgrades and improvements.

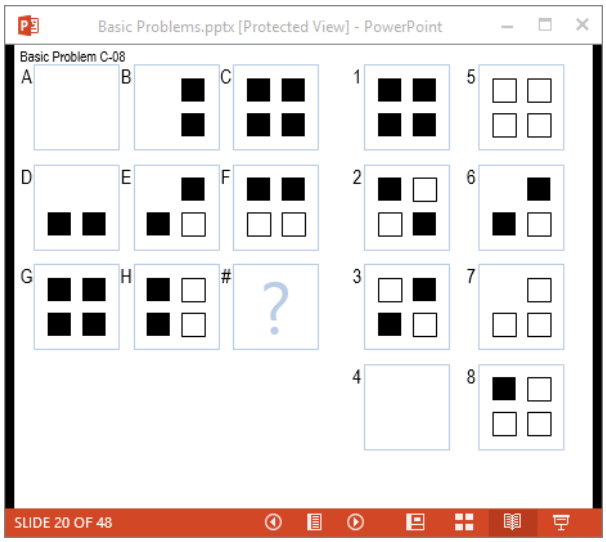
This was my object-matching implementation: I created a dictionary to store the object matches between figures that were made with the *matchObjects* method which inputted two figures to be compared, an attribute list for comparisons, and their relative weights. An object candidate list was created from the figure with more objects and these were iterated through against every object in the other figure. Matches were scored based on the following similar attributes and their relative weights. If a similar attribute was found, that object's score was incremented by the corresponding weight.

```
attributeList = ['size', 'shape', 'fill', 'angle', 'alignment', 'above', 'inside', 'left-of'] # list of attributes used to match objects
incrList = [10, 15, 5, 5, 4, 4, 4, 4] # corresponding attribute weights
```

These weights were in similar order to the transformation weights above. I reasoned that shape and size were the most important in matching, followed by fill then angle, and then positions.

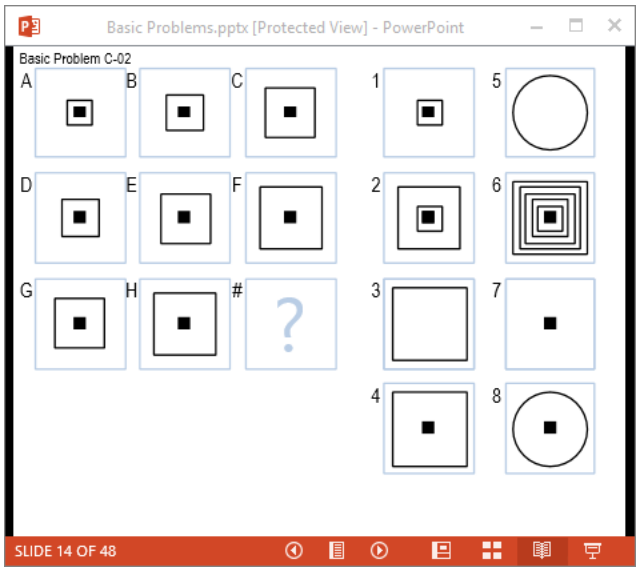
Once a match was made, that corresponding object was removed from the candidate list until all matches were made. The objects remaining in the list were assumed to be deleted or added but weren't used in the transformations comparisons. Once the object matching was in place, I then had to incorporate object matching into my existing transformation methods.

My agent ran into trouble when it attempted Problem C-08 (below) as it didn't know what to do with the empty figure in answer 4.



Therefore I had to set number of objects manually to zero so that the deletion/addition difference could be calculated between A C and A G and C I and GI

Since my agent was having trouble correctly answering C-02 (below) based on size transformations, I added size transformations.



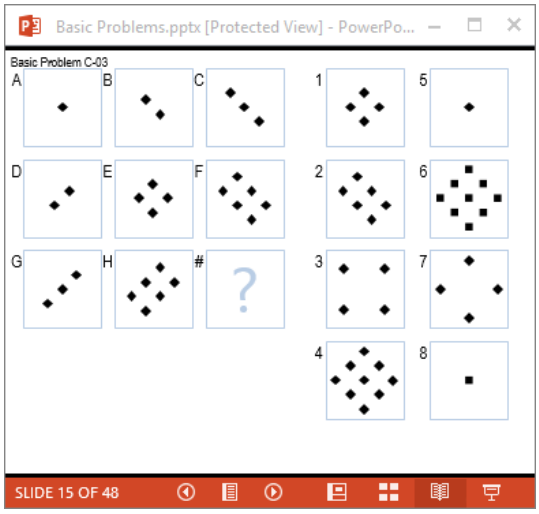
For its implementation, I first created a method where I converted sizes into integers, eg.

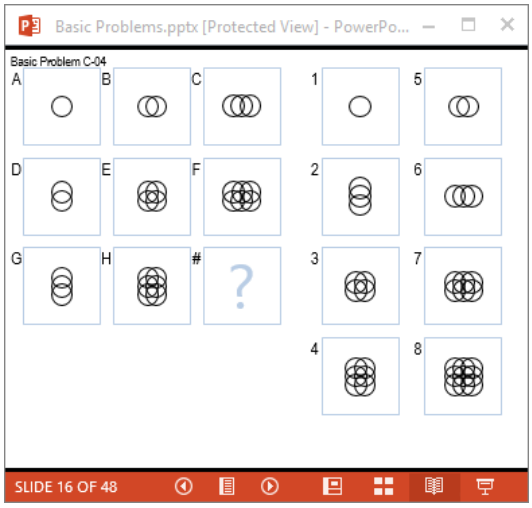
: `def convertSize(self, size):`

```
    ' converts object size to integer'
    if size == 'very small':
        return 1
    elif size == 'small':
        return 2
    elif size == 'medium':
        return 3
    elif size == 'large':
        return 4
    elif size == 'very large':
        return 5
    elif size == 'huge':
        return 6
```

Then I calculated the integer difference between the objects as a transformation calculation and then compare that number to the corresponding transformation. Eg. (A to B) compared to (C to D).

Once I added all the updates to my program described above, I was running into trouble with problems that add new objects to the frames, such as C-03 and C-04 (below)





I noticed that we could make a transform comparison that compares the multiplication from A to C versus G to I and similarly for A to G and C to I.

For example in both these problems the number of objects was multiplied by 3.

I first added a compensation for added objects similarly to the compensation for deleted objects.

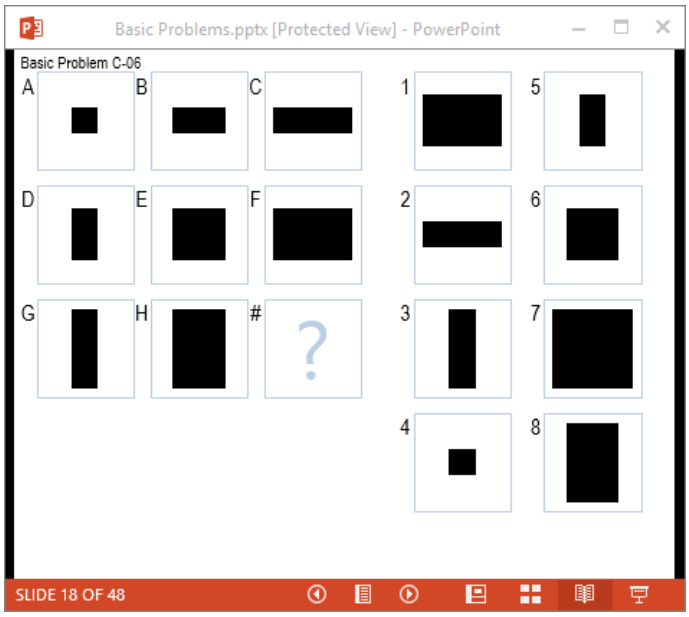
The compensation increment was:

tot += (ansLen - max(aLen, bLen, cLen)) * deleteCompIncr

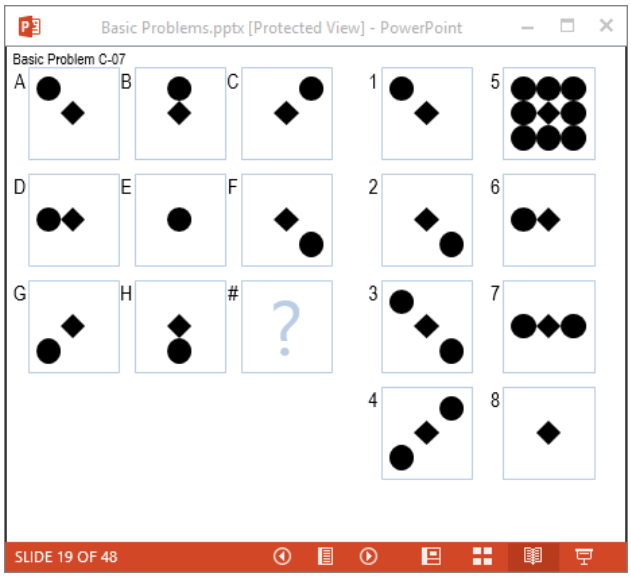
When adjusting the deleteCompIncr down to 20, I was able to correctly solve problems C3 and C4. For greater accuracy and generality, I then added the deletion multiple transformation and assigned it a weight of 2 * deleteCompIncr. This reinforced the correct answer in problems types 3 and 4.

At this point my AI agent was correctly answering 12/12 of the B problems and 6/12 of the C problems.

However, C-6 to 12 excluding 9 were answered incorrectly. So I set out to problem solve them one by one.

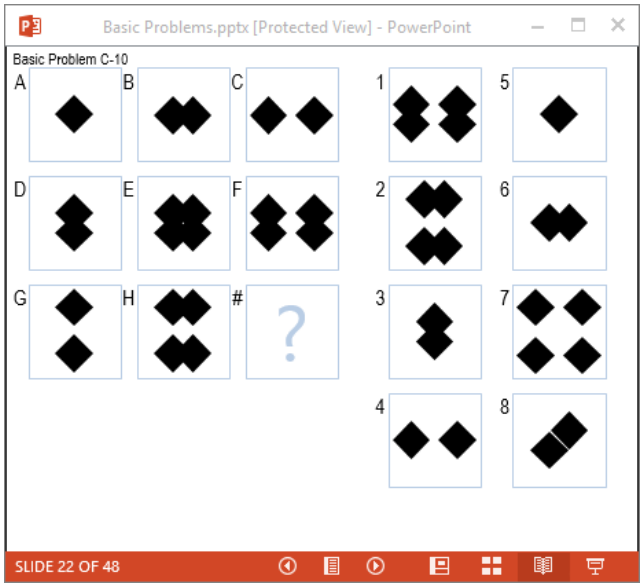


Problem C-06 (above) introduced the new problem of a square being stretched into a rectangle in each dimension. Therefore, if the shape was a rectangle it had a height and width property. If the shape was a square, I gave it an equivalent height and width value as the size. I then compared the transform across and down to determine the answer. This method correctly solved problem C-06.



Problem C-07 (above) was quite tricky. The correct answer of 2 looking almost symmetric.

For this I had to add a diagonal comparison. There was symmetries among the diagonals. I added a “right of” attribute to the object of that “left-of” corresponds to, and similarly “below” for “above”, and “outside” for “inside”. Unfortunately I ran out of time to properly make the transformation comparisons with these attributes.

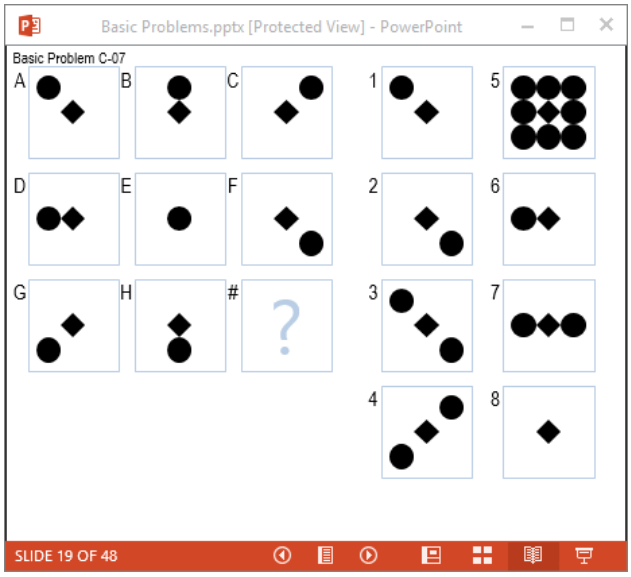


Problem C-10 (above) involved first cloning a diamond in A into 2 overlapping diamonds in B and then separating 2 overlapping shapes which move apart horizontally in C. The same happens in the vertical dimension moving downwards.

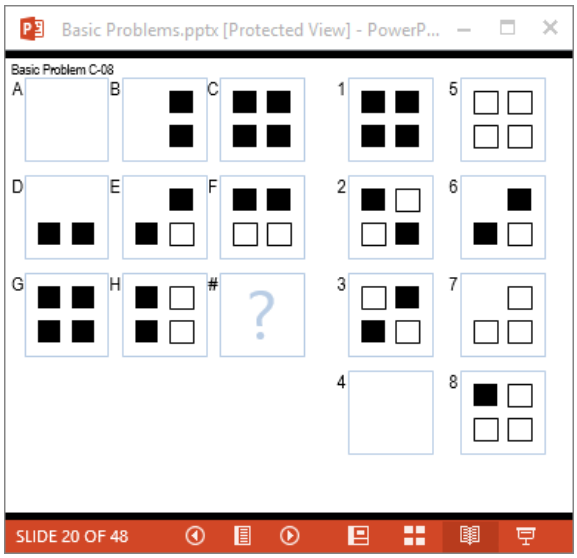
The objects of this problem had the new property of "overlaps". Therefore I had to make a new attribute of 'overlapsBool', an overlap Boolean of yes/no indicating whether an object overlaps in order to compare transformations. This worked successfully for C-10.

- *What mistakes does your agent make? Why does it make these mistakes? Could these mistakes be resolved within your agent's current approach, or are they fundamental problems with the way your agent approaches these problems?*

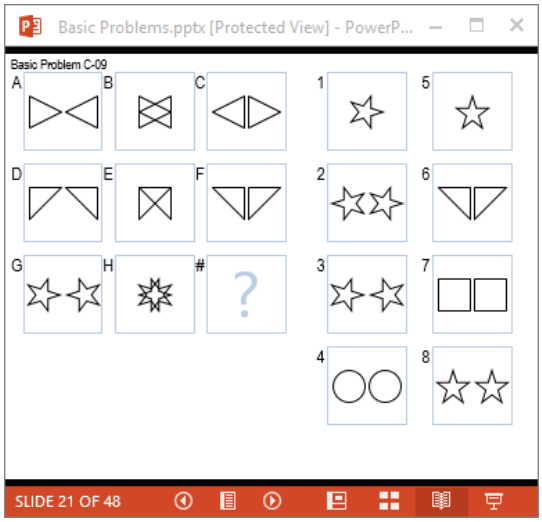
  My agent unfortunately makes some mistakes. It only answers problems C11 and 12 correctly 50% of the time with the random tie break, but it doesn't yet correctly answer problems C 7, 8, and 9. I believe all of these mistakes could be resolved within my agent's current approach as will be explained below.



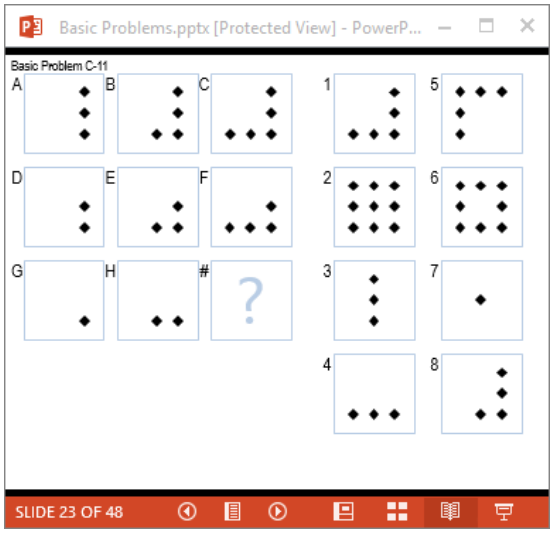
Problem C-07 (above) could easily be solved once I successfully implement the transformation comparisons of ‘above’, ‘inside’, and ‘left-of’ as described above. It already answers very close, answer 3 (score of 2677) instead of answer 2 (score of 2673)



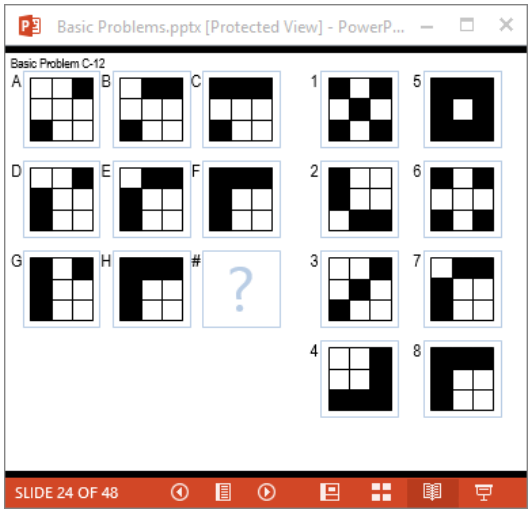
Problem C-08 (above) was answered very close as 1 instead of 5. An overlap transformation would have to be added in order to detect overlapping squares being unfilled.



Problem C-09 (above) only required the positional transformations to be implemented. Otherwise it chose very close 3, instead of 2.



Problem 11 (above) was sometimes correct based on a tie break between 3 and 4 which are very close. This would be solved by the spatial comparison of left-of and above.



Problem 12 (above) would also likely be solved by spatial left-of and above transformations.

What improvements could you make to your agent given unlimited time and resources? How would you implement those improvements?

Given unlimited time and resources, as explained above I would correctly implement my "above", "inside", and "left-of" transformation calculations in a similar way to my existing transformations.

I would also implement visual reasoning and determine an answer from a combination of the verbal and visual. I believe this would take the best from both worlds and would especially help for the symmetrical type problems and in recognizing "motion" of objects positions, additions/deletions, sizes, or fills in one direction as we move through the frames, as was the case of most of the C problems.

Another improvement would be to incorporate Learning by Recording Cases in order to adjust the weights which would be implemented as follows: If as a result my current weighing system my agent chose an incorrect response, it would adjust the weights to values that would cause the agent to choose the correct response (the agent retrieves after it submits its answer). This will hopefully aid the agent in making better choices in future problems. Finding the optimal parameters could achieved by use of an algorithm which randomly adjusts the different weight parameters and runs through and provides an answer to all of the test RPM problems which have solutions currently available in order to find the optimal parameter set. This could be done by randomly assigning weights and checking each time for the total number of correct answers and choosing the weights with the highest number of correct answers.

- *Would those improvements improve your agent's accuracy, efficiency, generality, or something else?*

These improvements would improve my agent's accuracy as there would be greater differences in the scores for answers as there would be more metrics to measure the best answer. Also the transformation weights would be more accurate.

It would also improve generality as more metrics and more accurate transformation weights would more accurately determine the correct answer over more varied problems than the ones given.

Efficiency would slightly go down as there would be more checks for transformations with every loop. However, this would be barely noticeable for our purposes as the agent runs in about 1 second. Optimizing the transformation weights could be computationally costly as many iterations may be necessary.

- *How well does your agent perform across multiple metrics? Accuracy is important, but what about efficiency? What about generality? Are there other metrics or scenarios under which you think your agent's performance would improve or suffer?*

My agent does reasonably well across multiple metrics, but sometimes gives the wrong answer based on lack of correctly implemented positional transformations of "left-of", 'above,' and 'inside' as explained above. This can hurt generality.

Efficiency is pretty fast for our purposes, taking around 1 to 2 seconds to compute. However, there are many double nested loops which may slow down the computation over large data sets.

Other scenarios such as ones with many shapes may cause my agent's performance to decrease slightly as there could be a higher chance of object mismatches.

- *Which reasoning method did you choose? Are you relying on verbal representations or visual? If you're using visual input, is your agent processing it into verbal representations for subsequent reasoning, or is it reasoning over the images themselves?*

I relied on verbal representations reasoning method as explained above.

- *Finally, what does the design and performance of your agent tell us about human cognition? Does your agent solve these problems like a human does? How is it similar, and how is it different? Has your agent's performance given you any insights into the way people solve these problems?*

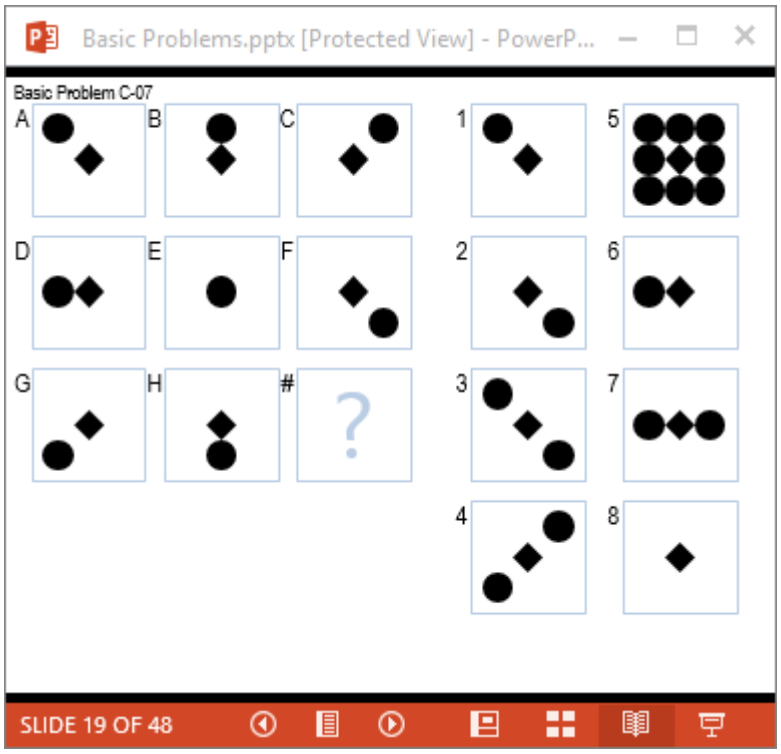
  The design and performance of my agent tell us the following about human cognition: That our minds make many computations, decisions, and interpretations without us being aware. Designing the agent forced me to break the transformations down to simple steps instead of just solving the figures by merely looking at the problem and deciding which answer looks correct.

  It also tells us that humans and AI agents may think differently. My agent somewhat solves the problem like a human does.

  It is different in that it makes a step by step comparison between the figures and its neighbors to see what transformations are made in order to find the best solution. Eg. It

first checks shape change, then fill, then rotation, etc. However, it is more similar to people in more complicated problems where a step by step comparison is required.

It is different in that it randomly loops through all objects in the figure and compares them to its corresponding object. Also, object matching was very difficult to program in an agent while it's very easy for a human after a quick scan of the figures to look for symmetry or object motion, or patterns. It appears that humans are much better able to see the big picture of the figures. For example, in problem C-07 (below), answer 2 looks like it fits in the best with symmetry even though F is out of place, which might trick the AI agent.



My agent's performance has given me lots of insights into the way people solve these problems. One is that people easily see how the answer fits into the big picture. Humans can more easily correspond objects based on quickly seeing shape, fill, angle, position

across figures. Also humans can make multiple transformations (eg. shape and fill) at the same time. Conversely, the AI agent has to make explicit multiple transformation comparisons in a step by step fashion in order to correctly choose the correct answer, shape, fill and angle. However, when more transformations are required for more difficult RPM problems, an AI agent has the advantage that it more explicitly breaks down the transformations into steps and computes those steps accordingly, providing a quicker, more systematic and perhaps more accurate answer than a human. Humans will likely have to use this thinking deliberately, in a similar fashion to the AI agent, in order to choose the correct answer.