In this paper, I will present my project 3 reflection by directly answering the questions in the reflection part of the project. You might notice that I will elaborate on some answers more than other. In some cases, I will also be addressing partial answers to a specific question under a different question if the ideas are related to the answer of the other question.

- **How does your agent reason over the problems it receives? What is its overall problem-solving process? Did you take any risks in the design of your agent, and did those risks pay off?**

The approach I am using is to look at each problem and try to address it while keeping two things in mind: generality and accuracy. I had to ensure that I do not design an agent that over fits the solution to get the correct answer but also it should not develop too general solutions that result in picking the wrong answer or a less correct answer. I started by looking at each question to know what **transformations/relationships** I need my agent to handle and concluded that the agent should be able to handle the following order: **logical and, exclusive nor, shifted logical and, no transformation, object addition, incremental addition or deletion, logical or, and logical exclusive or**. The first risk I had to take is to assume that these are the only transformations my agent is expected to handle. Since I only used the basic sets D, E to construct my agent, I only had a small set of 3x3 problems to test this assumption on which turned out to get a decent score of around 91% with (21) problems solved with a confidence of '1'. After identifying the possible transformations, it was also important to see the complexity of the solutions and similarity between them to determine if any attribute can or cannot be overlooked. So at a high level, I start by checking the possible transformations mentioned earlier between rows (ABC, DEF) of frames and if for a given transformation no relation between rows was found that

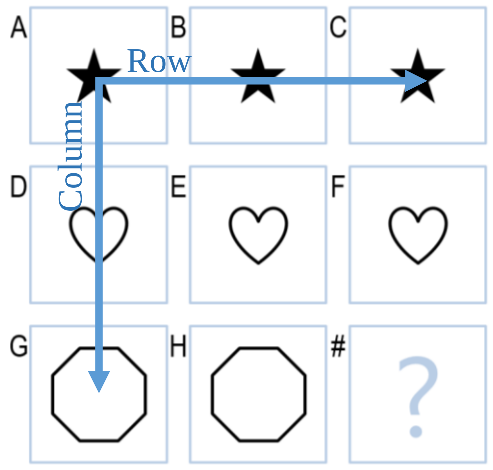


***figure1**: transformation check*

satisfies the transformation, I check the same transformation between columns (ADG, BEH) then diagonally if columns have failed.

In the case a relationship was found between rows, it will be extended iteratively between G, H and the possible answers and check for an answer that satisfies that relationship and if an answer is found, it is returned alone with confidence of 1 for some transformations while other transformations check all possible solutions and return the answer with all potential solutions marked with a value inversely proportional to their number. Similarly, if going from A to B fails, the same strategy will be applied between A & C where B will be paired with the possible answers. In the case the tested relationship fails to yield an answer between rows or columns, the next relationship in the order mentioned above will be tested. With this approach lies a risk that the possible answers have multiple answers each is based on a certain relationship and only the answer that has the relationship that is higher in the order of relationships will be selected regardless of the correctness of the other possible answer(s) for those transformations that only return one answer. The reason I decided to overlook this fact was the nature of the problems the transformation was handling in basic problem sets D and E have mostly definitive possible answers where the correct answer clearly stands. From above, it is clearly shown that my agent assumes horizontal (row) or vertical (column) or diagonal relationships depending on the transformation it is examining. Similarly, this also extends to returned answers where depending on the transformation examined either a single answer is returned or a list of possible answers.

- **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?**

  The way my agent select an answer is by applying a transformation function on both tested frames. I have written functions for each of the aforementioned transformations (logical and, exclusive nor, shifted logical and, no transformation, object addition, incremental addition or deletion, logical or, and logical exclusive or) where each of the functions take two frames as input for example (A & B) and look for certain attributes in both frames based on the transformation and determines whether this transformation is applicable or not. For example, the no transformation function compares frames A and B visually and calculates the Jaccard Index[1] between the two. If the index is higher than a threshold (t), then it is considered a match. The threshold has been identified experimentally and chosen depending on the transformation type being examined. To calculate the Jaccard Index, I find the black pixel density of the intersection of the two frames divided by the black pixel density of the two frames. This results in a number between 0.0 to 1.0 where 1.0 is a match and 0.0 is a mismatch so closer to one is closer to a match.

Previously, I used the rms error as a metric to compare images but upon experimentation, I found out that the Jaccard Index with different thresholds for each transformation yields much better results.

Due to the nature of some simple problems in basic set D and E, I took the risk of returning the first answer found when examining some transformations so the agent always returns 1 if a solution was found by checking for possible relationships and extending them to check for a solution. In other transformations however, I found it more accurate to check all solutions and rate them then choose the answers that pass the test and return them with a value inversely proportional to the number of possible answers. It worth noting that the rating of each potential answers is binary so it either passes or not and not continuous.

- **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?**

  One of the mistakes my agent had was related to the order of relationships. Initially, I checked for the order of the relationships in a order different than that mentioned earlier. I noticed however that agent is failing in some of the logical and problems as shown in figure 2 and was returning an incorrect answer; it was doing so due to considering the problem as a different type of transformation problem without even attempting the logical solution.
  This was fixed by changing the order of the checked transformations so it wasn't a fundamental problem with my approach rather an overlooked mistake.

  Another mistake my agent makes was having generalized solutions for some of the problems rather than a specific solution. This occurs for example in problem D-12 where my agent mistakes answer 6 as a potential answer along with answer choice 3 as shown in figure 3. The reason this occurs is because when solving the problem I examine the change in black density as an incremental growth but it seems that density in choices 3 and 6 are very close. The reason why this error might have occurred was that I did not want to over fit my functions to only solve a subset of problems with high accuracy while failing in others and by assessing the tradeoff between the two, a more general solution seemed more applicable with my overall approach so I decided to keep it as is since the answer is still logical but very specific. So in this case, this is a fundamental problem in my approach but is intentional to maintain generality.

***figure2**: problem E-01*

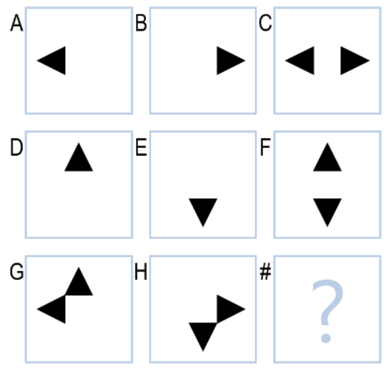
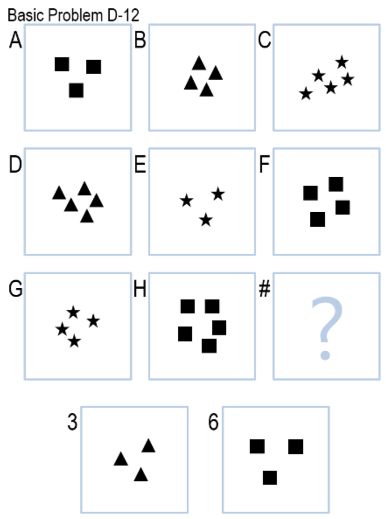


***figure3**: problem D-12*

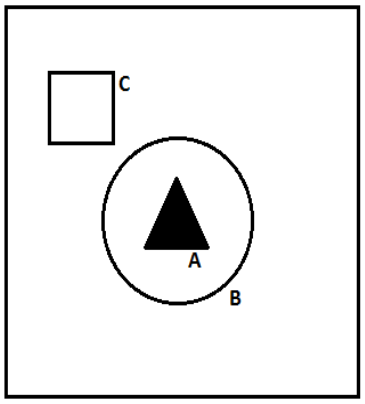


- **What improvements could you make to your agent given unlimited time and resources? How would you implement those improvements? Would those improvements improve your agent's accuracy, efficiency, generality, or something else?**

  In general, I would want to employ a more complex approach that can solve problems beyond simple problem sets D and E which I talked about in the previous assignments but did not have time to implement in this project. This approach builds a matrix for each frame similar to SLAM that relates all objects in the frame together. The goal of this matrix is to assign a unique ID to each object that will make it easy to compare frames. For example let us assume we only have 3 object in a frame and call them A, B, and C as shown in figure 3 below.

  Also for simplicity, let us assume we only have the following attributes: inside, below, fill and un-fill where the size is not an attribute and all objects are of the same size. We will then create a matrix that relates these objects based on figure 3 in a manner similar to graph slam. To do so, we first need to assign unique values to each attribute. For example, the following unique values can be assigned to each attribute:

***figure4**: Raven Figure*



  - 1 for inside
  - 10 for fill
  - 100 for unfill
  - 1000 for below

  The matrix that relates the objects will look something like:

	A	B	C		sum (id)	id_identifier
A	10	1	1000	=	1011	't'
B	0	100	1000	=	1100	'c'
C	0	0	100	=	100	's'

By assigning unique identifiers that are 10 apart we can produce a unique id for each object that can be added (this holds for frames with a small number of items, alternatively unit vectors can be used to denote relations for frames with large number of objects and relations). It can be seen here that with this method, we can assign unique ids to objects that make object comparison much easier. This technique also assigns weights or importance to attributes. For example, we can think of un-fill more important than inside so if something changed from fill to un-fill it will count more than something changed from inside to not inside.

After obtaining the unique ids for each object, these objects can be easily compared with other objects in a different frame and try to establish the relationship on the object level rather than the frame level by seeing how close two ids are. Changes can also be deduced by examining the ids since each magnitude of 10 is certain attribute and by subtracting two objects, we can find difference/relationship that occurred. For example object A's id is '1011' which reads: 'below an object, filled and inside an object' and if the id changed to '1001' we can immediately see that the object is no longer filled. Another example would be if 'C' changed to enclose both 'A' and 'B' then the id of A will be '0012' which is 'filled and inside two objects'. The last part of the id, is the id identifier which basically is the object type and this is especially important to keep track of objects when you have multiple objects in each frame so before looking at ids we can check the identifiers of both objects to check if they are of the same kind. The main issue with this implementation is the efficiency as this will result in a large search tree that starts with frame level, object level comparison then a set of answers to each branch of the tree.

This approach clearly increases generality and probably accuracy as well as the ability to tackle more challenging problems but will be definitely be less efficient than the original approach. It will also require me to translate visual problems to verbal problems.

- **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?**

  In addressing basic problem sets D and E, I would say that the agent handled those problems both efficiently (relatively speaking) and accurately (91%) but not as efficiently as the previous sets B and C. This is mainly due to just using a visual approach that check for multi-variants of each frame as well as sets D and E being more difficult than C and B. The average runtime for my agent is around 6.75 seconds per problem which I think could be improved by using matrix operations rather than loops in my calculation. Another metric is confidence, where I think the agent did a good job in being 100% confident in 21/24 problems and 0.5, 0.25, 0.125 for the other three accordingly. The reason it lacked confidence in the three problems might be because I wanted to avoid overfitting my solution to this set. As for generality, I would say that it could solve problems similar in difficulty to those in problem basic sets D and E. However, if the level of difficulty increased where multiple transformations are taking place between frames or if some objects are for example rotated and others are deleted, I would assume that the agent will not produce reliable solutions. As a matter of fact, the agent performed quite poorly in challenge problems D & E due to them being different in nature than basic problems D & E and it would need more knowledge about them in order to produce accurate results.

- **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?**

  Parts of this question have already been addressed earlier but to summarize, my agent employed only a visual reasoning approach when solving problems and comparing figures. When I used the visual approach, I did not process images into verbal representations. Instead, I used the used the previously mentioned Jaccard Index to check for similarities or differences between images based on a formula found here:

  https://en.wikipedia.org/wiki/Jaccard_index

  But I found out that in many cases there are some residuals when comparing maybe due to misalignment so a threshold was used as mentioned earlier.

- **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?**

While designing the agent, I came to realize that the agent differs than humans in the sense that it has a structured pre-determined and rigid approach. While a human (my self) have a more free approach that is drawn to certain things such as symmetry for example and do not follow an approach as structured as the agent's approach. Upon examining the problems in this project or other similar problems, I noticed that there are salient attributes that humans are drawn to but agents might overlook or ignore depending on the way they are programmed. Another fundamental difference is adaptability, where humans are more adaptable than agents when solving these problems and can establish a variety of complex relationships that an agent would fail to establish. A point worth mentioning is that maybe a human is more adaptable than an agent due to the vast knowledge that the human acquired throughout the years and maybe an infant who was not exposed to that amount of knowledge and could not understand some of the relationships would perform worse than an agent so we are similar in the sense that both humans and agents apply previously acquired knowledge in solving these problems.