Reflection on Project 3: Introduction

For projects 1 and 2, I solved Ravens Progressive Matrices (RPM) utilizing verbal data. For project 3, only the images were provided, requiring a visual approach to solving RPM. It is assumed the reader of this reflection has seen RPMs, therefore, I will not provide a definition for RPM[i] in this reflection.

Although the approach for evaluating RPM data was significantly modified for project 3, I was able to reuse the scoring and selection methodology from the previous projects. This was helpful for utilizing my time to focus on the evaluation; however, it might have had a negative impact in scoring. I will explore this in the subsequent sections of this reflection.

Since the most significant modifications were to my Agent's (visual) evaluation of the data, this will be the focus for this reflection. My overall approach was to use a single Agent with three production rules: ***MergeImage***, ***NumPyStats*** and ***NoMergePix***. Of the three, ***MergeImage*** seemed to be a unique and risky rule that will get most of the attention in this reflection. I use the word unique based on postings to the piazza message board for our class. It appears that the methods many students used to solve RPMs involved some form of image decomposition. That is, an image is dissected to identify edges, shapes or transpositions. I took the opposite approach, and merged all images in a row or column to identify colors in the Red, Green, Blue (RGB) color spectrum. Since the RPM images are all composed of black (RGB = [0, 0, 0]) and white (RGB = [255, 255, 255]) pixels, this meant that after merging images, I was analyzing colors in the grey scale (RGB = [x, x, x], where x != 0 or x != 255). This process proved extremely efficient in terms of code complexity and performance. I have one Agent.py file that consists of 636 lines of code, of which 63 lines are comments. Depending on the number of production rules that are applied, each problem takes between one and nine seconds to return an answer. However, although the Agent proved very successful in solving problems in Basic D, it was not as strong in problem sets C and E. Lastly, I will reflect on the cognitive properties my Agent exhibits in solving RPMs.

Problem Solving Approach

As I began to consider a strategy for solving the RPMs visually, I started with the Basic B problem set. I found that comparing just the black pixels between two images was sufficient to correctly solve 10/12 Basic B problems. With some adjustments to account for symmetry, I was able to get 12/12 Basic B. However, this success was not maintained in problem sets with three images per row/column. As I considered alternatives for data evaluation, I started merging two images and noticed an interesting pattern of grey. For projects 1 and 2, I evaluated and made decisions by comparing two figures, so it seemed logical to merge two images: AB, CD, AD for rows, (likewise for columns) for the evaluation and selection process. This experiment was still not scoring better than forty percent correct on problem sets B, C and D. I decided to take a risk and evaluate an entire row, column or diagonal by merging all the images together.

```
validCompRow3x3 = (['ABC', 'DEF',
                    'GH1', 'GH2', 'GH3', 'GH4', 'GH5', 'GH6', 'GH7', 'GH8'])

validCompCol3x3 = (['ADG', 'BEH',
                    'CF1', 'CF2', 'CF3', 'CF4', 'CF5', 'CF6', 'CF7', 'CF8'])

validCompDiag3x3 = (['BFG', 'CDH',
                     'AE1', 'AE2', 'AE3', 'AE4', 'AE5', 'AE6', 'AE7', 'AE8'])
```

When three images are merged, some very interesting shades of grey appear in the data. The ***MergeImage*** rules merge images, identify and count the pixels from the merged image.

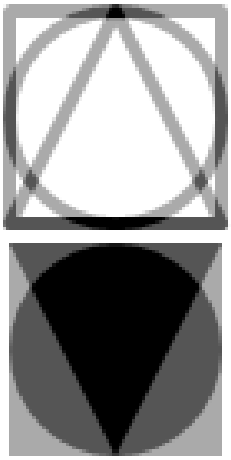
```
# Major spectrum, black, white
if ( value[:3] == (0, 0, 0) ):
      black += 1.0
if ( value[:3] == (255, 255, 255) ):
      white += 1.0

# Major 'Grey' spectrum
if ( value[:3] == (42, 42, 42) ):
      grey01 += 1.0
if ( value[:3] == (85, 85, 85) ):
      grey02 += 1.0
if ( value[:3] == (127, 127, 127) ):
      grey03 += 1.0
if ( value[:3] == (170, 170, 170) ):
      grey04 += 1.0
if ( value[:3] == (212, 212, 212) ):
      grey05 += 1.0

# Minor 'Grey' spectrum
if ( value[:3] == (69, 69, 69) ):
      grey06 += 1.0
if ( value[:3] == (111, 111, 111) ):
      grey08 += 1.0
if ( value[:3] == (154, 154, 154) ):
      grey09 += 1.0
if ( value[:3] == (196, 196, 196) ):
      grey07 += 1.0
if ( value[:3] == (239, 239, 239) ):
      grey10 += 1.0
```

This is an example of how my ***MergeImage*** Agent evaluates the grey colors from Basic Problem E-09, column CF6:

```
CF6 {'grey10': 36.0, 'grey07': 32.0,
'grey06': 2.0, 'grey05': 7.0, 'grey04':
1583.0, 'grey03': 10.0, 'grey02': 1929.0,
'grey01': 8.0, 'black': 1753.0, 'white':
27534.0, 'grey09': 9.0, 'grey08': 11.0}
```



There are two other production rules:

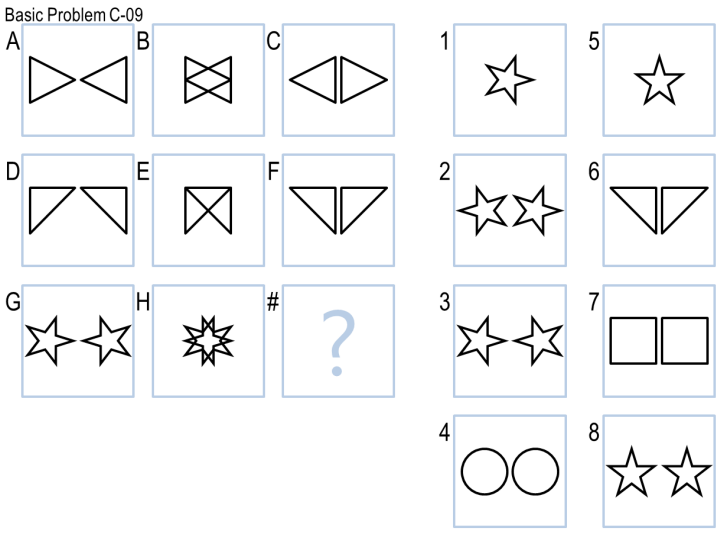
- ***NumPyStats***: Performs basin NumPY operations on merged images.
- ***NoMergePix***: Counts and compares pixels from three images in a row, column or diagonal.

The focus will be on ***MergeImage*** as these rules were used for all problem sets and provided the most weight to the decision making process. Problem set D was the only problem set that used all three production rules, however, my Agent scored 10/12 using only the ***MergeImage*** rules for D. ***NumPyStats*** was used only for Problem set C, while Problem set E used only the ***MergeImage*** rules. I used a conditional based on the Problem set for selecting the production rules.

Let's examine the ***MergeImage*** more closely by looking at some examples. Three examples are chosen, one from each problem set: C, D and E. I selected these problems based on the following criteria:

- Problem C-09 has transposition, or rotation and images without fill.
- Problem D-07 has symmetry, and images with and without fill.
- Basic Problem E-11 is not symmetric, nor does it exhibit transposition.

Example 1: Basic Problem C-09



This problem shows how ***MergeImage*** solves a problem that has transposition and no fill in the images. The ***NumPyStats*** rules are applied, but provide confirmation to the correct solution, but only minimally. The ***MergeImage*** rules do a good job in choosing the answer and provide most of the weight. The ***NoMergePix*** rules are not used.

The merged images for the row and column are provided. Looking at the merged image for the correct answer (2) and the next best choice (4), it is difficult for me to distinguish the answer.

Results from debug:

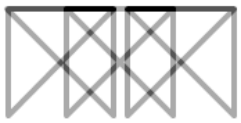
MergeImage [9686.0, **8035.0**, 12568.0, 8934.0, 9076.0, 15342.0, 12984.0, 10194.0]
NoMergePix [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
NumPyStats [5768, **5764**, 5756, 5618, 6302, 8368, 7686, 5792]

Tot [15454.0, **13799.0**, 18324.0, 14552.0, 15378.0, 23710.0, 20670.0, 15986.0]

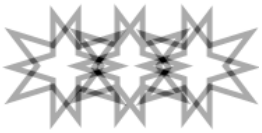
Ans [0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

ABC

DEF



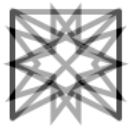
GH2



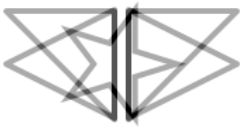
GH4

ADG

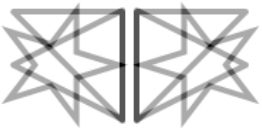
BEH



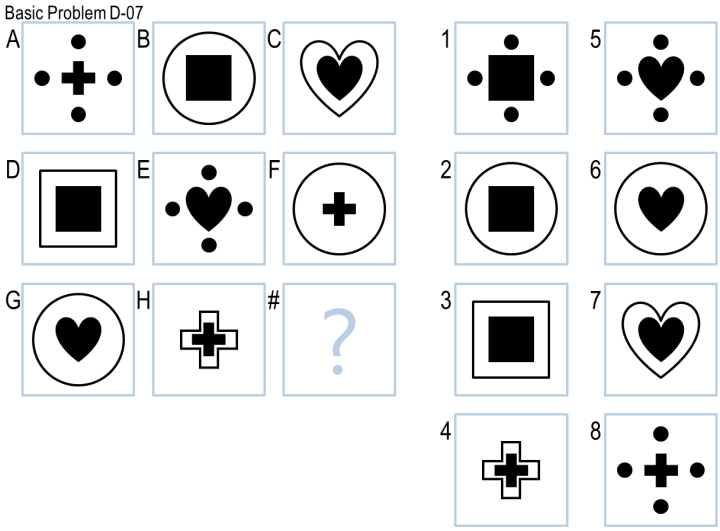
CF1



CF3



Example 2: Basic Problem D-07



This problem shows how ***MergeImage*** solves a problem that has symmetry and images with and without fill. All the production rules are applied to solve this problem. The rules for both ***MergeImage*** and ***NoMergePix*** strongly identify the correct answer. In this case the rules for ***NumPyStats*** did not identify the correct answer, but not by much, and not enough to offset the choice of the other two production rules.

The merged images for the row and column are provided. Looking at the merged image for the correct answer (1) and the next best choice (3), I can see that four small, light grey circles appear to identify the correct answer.

Results from debug:

MergeImage [**14649.0**, 36215.0, 16493.0, 36989.0, 32237.0, 46051.0, 40343.0, 32139.0]
NoMergePix [**4858.0**, 5894.0, 5846.0, 24326.0, 22758.0, 26190.0, 26890.0, 26890.0]
NumPyStats [**10288**, 11838, 9690, 22350, 17136, 12732, 11760, 27438]

Tot [**29795.0**, 53947.0, 32029.0, 83665.0, 72131.0, 84973.0, 78993.0, 86467.0]

Ans [1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

ABC

DEF

GH1

GH3

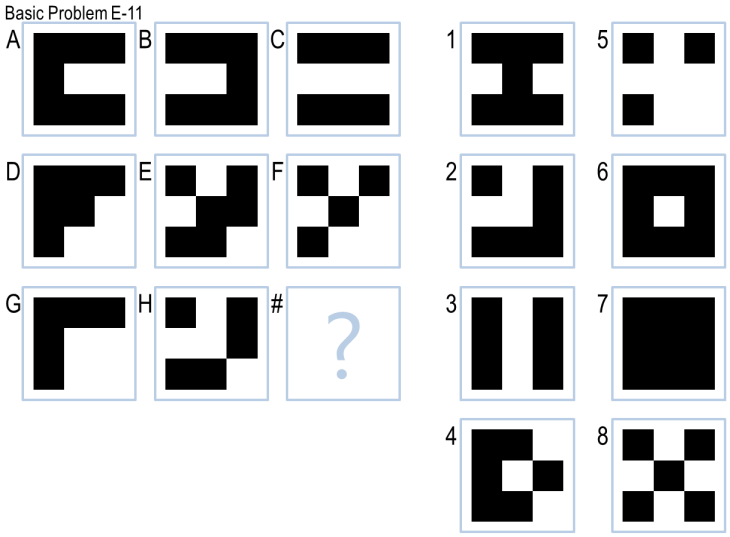
ADG

BEH

CF1

CF3

Example 3: Basic Problem E-11



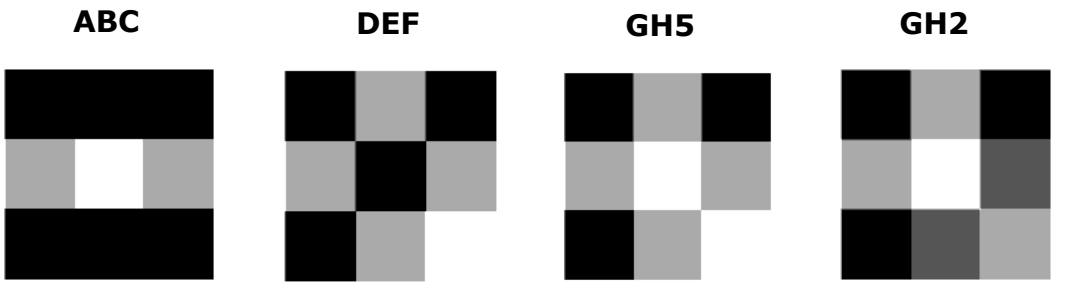
This problem shows how ***MergeImage*** solves a problem that is not symmetric, nor does it exhibit transposition. For this problem, the ***MergeImage*** rules are the only rules that are applied.

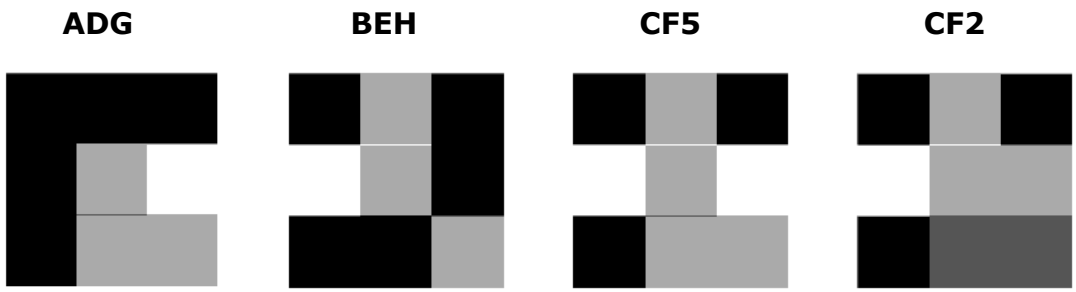
The merged images for the row and column are provided. Looking at the merged image for the correct answer (5) and the next best choice (2), it appears that a medium shade of grey appear to distinguish the correct answer.

Results from debug:

```
MergeImage [109026.0, 62874.0, 82676.0, 127418.0, 46822.0, 108336.0, 129962.0, 77933.0]
NoMergePix [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
NumPyStats [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Tot [109026.0, 62874.0, 82676.0, 127418.0, 46822.0, 108336.0, 129962.0, 77933.0]
Ans [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0]
```





Evaluation and Selection Process:

I was able to reuse the evaluation and selection criteria from the previous project for project 3. I will not go into great detail on these topics. If additional detail is desired, please refer to my reflection for projects 1 or 2.

I evaluate by computing the Euclidian distance as follows:

- Gather data from the production rules: ***MergeImage***, ***NumPyStats***, ***NoMergePix***.
- Use the data obtained from the rules to compute the Euclidian distance from rows with three images to a row with two images and a numbered choice.
  - 'ABC', 'DEF' to 'GH1', 'GH2', 'GH3', 'GH4', 'GH5', 'GH6', 'GH7', 'GH8'
- Do the same for columns and diagonals.
  - 'ADG', 'BEH' to 'CF1', 'CF2', 'CF3', 'CF4', 'CF5', 'CF6', 'CF7', 'CF8'
  - 'BFG', 'CDH' to 'AE1', 'AE2', 'AE3', 'AE4', 'AE5', 'AE6', 'AE7', 'AE8'
- Produce vectors of the Euclidian distance corresponding to each potential answer [1 - 8] from data obtained from the production rules. See results from debug in Examples 1,2 and 3.
- Sum the vectors of data from the production rules.

I select answers based on the minimum value of the sum of the three vectors of data from the production rules. I choose the minimum value and perform no filtering or elimination of choices. If there are multiple minimum values, they are weighted equally. I do not perform any additional calculation for breaking ties.

```
maxScore = max(normListOfConfidence)
for x in range(len(normListOfConfidence)):
      if normListOfConfidence[x] == maxScore:
            normListOfConfidence[x] = 1.0
      else:
            normListOfConfidence[x] = 0.0
```

I then normalize the list: `normList = [abs(0 - (x/sumList)) for x in f]`

Shortfalls and Improvements

Although my Agent was very successful in identifying the correct answers for Problem set D. It was not as good on problem sets C and E. Initially, using the production rules from ***MergeImage*** was scoring very well on Problem set D, but only 5 correct on Problem set C and 6 on Problem set E. I added a diagonal comparison to improve to 8 correct on problem sets C and E. However, this negatively impacted how my Agent scored on Problem set D. I had to add additional rules: ***NumPyStats***, ***NoMergePix*** to compensate. This runs the risk of over-fitting the data for a particular problem set. Looking at how my Agent scored on the Challenge Problem sets, it points to the potential of over fitting, or at least some limitation to how my Agent will perform against unknown problems.

Given more time, I would modify how my Agent evaluates answers. I would not compare within rows, columns, diagonals, but make decisions based on the values between rows, columns, diagonals. I would also use this data to filter out choices that are obviously not correct, instead of considering all choices equally.

I would also add in some computations to account for rotation. I did try this, but on merged images. It did not improve my scores, and negatively impacted performance in terms of time. I would like to revisit this by computing on the images without merging.

Performance

My Agent solved 10/12 Basic B, 8/12 Basic C, 11/12 Basic D and 8/12 Basic E. I also included performance on the Challenge problem sets. The runtime is dependent on the number of production rules that are called. The problems in set E and C run very fast, ~ 2 seconds , while the problems in D take ~8 seconds.

Problem	Score	Time	Problem	Score	Time
Basic B-01	1	1430.00	Basic C-01	1	2460.00
Basic B-02	1	1370.00	Basic C-02	0	2480.00
Basic B-03	1	1380.00	Basic C-03	0	2470.00
Basic B-04	1	1350.00	Basic C-04	0	2490.00
Basic B-05	1	1440.00	Basic C-05	1	2450.00
Basic B-06	0	1390.00	Basic C-06	0	2500.00
Basic B-07	1	1380.00	Basic C-07	1	2490.00
Basic B-08	1	1450.00	Basic C-08	1	2480.00
Basic B-09	0	1390.00	Basic C-09	1	2560.00
Basic B-10	1	1380.00	Basic C-10	1	2560.00
Basic B-11	1	1360.00	Basic C-11	1	2460.00
Basic B-12	1	1460.00	Basic C-12	1	2460.00
TOTAL	**10**	**16.78**	**TOTAL**	**8**	**29.86**

Problem	Score	Time	Problem	Score	Time
Basic D-01	1	8050.00	Basic E-01	1	2010.00
Basic D-02	1	7990.00	Basic E-02	0	2010.00
Basic D-03	1	7960.00	Basic E-03	1	1950.00
Basic D-04	1	8050.00	Basic E-04	0	2030.00
Basic D-05	1	8080.00	Basic E-05	1	2030.00
Basic D-06	1	7970.00	Basic E-06	1	2010.00
Basic D-07	1	8090.00	Basic E-07	1	2020.00
Basic D-08	1	8060.00	Basic E-08	1	2030.00
Basic D-09	1	8070.00	Basic E-09	0	2030.00
Basic D-10	1	8060.00	Basic E-10	1	2030.00
Basic D-11	1	8070.00	Basic E-11	1	2010.00
Basic D-12	0	7990.00	Basic E-12	0	2020.00
TOTAL	**11**	**96.44**	**TOTAL**	**8**	**24.18**

Problem	Score	Time	Problem	Score	Time
Challenge B-01	0	1390.00	Challenge C-01	1	2580.00
Challenge B-02	0	1360.00	Challenge C-02	0	2460.00
Challenge B-03	0	1390.00	Challenge C-03	0	2570.00
Challenge B-04	0	1460.00	Challenge C-04	1	2490.00
Challenge B-05	0	1390.00	Challenge C-05	0	2460.00
Challenge B-06	1	1450.00	Challenge C-06	0	2560.00
Challenge B-07	1	1470.00	Challenge C-07	0	2580.00
Challenge B-08	0	1380.00	Challenge C-08	0	2600.00
Challenge B-09	1	1470.00	Challenge C-09	1	2490.00
Challenge B-10	0	1380.00	Challenge C-10	0	2500.00
Challenge B-11	1	1370.00	Challenge C-11	0	2560.00
Challenge B-12	0	1370.00	Challenge C-12	1	2470.00
TOTAL	**4**	**16.88**	**TOTAL**	**4**	**30.32**

Problem	Score	Time	Problem	Score	Time
Challenge D-01	0	7960.00	Challenge E-01	1	2000.00
Challenge D-02	0	8060.00	Challenge E-02	0	2020.00
Challenge D-03	0	8060.00	Challenge E-03	0	2020.00
Challenge D-04	0	8060.00	Challenge E-04	1	2030.00
Challenge D-05	0	8080.00	Challenge E-05	0	2030.00
Challenge D-06	1	8000.00	Challenge E-06	0	2010.00
Challenge D-07	0	8070.00	Challenge E-07	0	1940.00
Challenge D-08	1	8060.00	Challenge E-08	0	2030.00
Challenge D-09	0	8080.00	Challenge E-09	0	2020.00
Challenge D-10	0	8090.00	Challenge E-10	0	2020.00
Challenge D-11	1	8060.00	Challenge E-11	0	2020.00
Challenge D-12	0	8070.00	Challenge E-12	0	2030.00
TOTAL	**3**	**96.65**	**TOTAL**	**2**	**24.17**

Reflection on Human Cognition

I found it interesting that my Agent performed with mediocrity, and with similar results for Problem sets C and E, but excelled at problems in D. Looking at the problem sets, I can see o real differences, but my Agent did. The problems were quite difficult for a human to address. I gave these Problem sets B, C and D to six colleagues to gather additional insight on human cognition for solving RPM. It turns out that my human volunteers solved each problem set equally, ~10/12 for each set. However, they took an average of ~11 minutes to solve each set. My Agent outperformed on time, and was close in accuracy to my subset of human problem solvers.

Each project gave me additional perspective on considering the relationship between human cognition and Knowledge Based Artificial Intelligence. From the semantic networks and frames that I utilized for project 1 and 2, to the production rules and visual analysis required for project 3, I was astounded at time on how

my Agent performed. There were instances where the problem was quite difficult for me to solve, however, my Agent was able to solve it in seconds. There were other cases where the problem was trivial, but my Agent failed to identify a correct solution. I believe this points to an incomplete set of KBAI methods that I incorporated for solving problems. I was only able to touch on the very basic KBAI concepts, and was not able to fully develop the more advances concepts in KBAI.

I'd like there to be a project 4, so I can continue the progress. I think that after a small break, I will resume this quest.

[i] https://en.wikipedia.org/wiki/Raven%27s_Progressive_Matrices