Li (Lily) Xu (lxu321@gatech.edu)

Project 3 Reflection

---

1. Introduction

Raven's progressive matrices was first published in 1938 and designed to measure one's mental ability to reason by analogy and form perceptual relations. RPMs are composed of 60 multiple choice questions, 5 sets with different difficulty levels, each contains 12 questions. The problems usually contain several panels of pictures with one missing. The test taker is supposed to pick up the best match of this missing piece from the provided candidates of answers. Since it is not verbal, RPMs can be used with persons from child to adult in any language. In this class, we will assess the algorithms and reasoning of our artificial intelligent agents by examining their performance against the RPM problems.

There are many potential difficulties about how to solve the RPM problems with an AI agent. For example, first of all, how does the agent start to understand a question? How can it recognize a circle, a triangle or any other element? Secondly, how does an agent build the concept of "transformation" or in another word, how does the agent decide on the changes and make correlations between elements before and after these changes? Moreover, how does an agent compare and rank different types of transformations and how does it find the best choice, if there is no perfect match? Here, in this last part of the project, I modified my AI agent to try to address the above-mentioned obstacles and solve RPM problems exclusively based on visual presentations.

2. A brief summary of the agent

- **Data structure**

First a quick summary of code structure in this project: The agent starts from the main method in RavensProject.java and reads in all the RPM problems. These problems are then stored in ProblemSet.java, which is composed of a list of Raven problems. Each Raven Problem is represented by RavensProblem.java, which consists of its problem name, problem type (such as “2x2” or “3x3”) and a HashMap of RavenFigures. Each RavenFigure is composed of a figure name (“A”-“H” or “1”-“8”) and the path of the visual file. The class RavenFigure also contains a HashMap describing attributes of Raven objects, but this will be ignored as Project 3 only use visual method.

With the path of a .pgn file, the program reads in the visual presentation of a RavenFigure:

```
RavensFigure A = problem.getFigures().get("A");
BufferedImage figureAImage = ImageIO.read(new File(A.getVisual()));
```

Then I implemented a method called `imageToArrayRGB` () to covert data stored in the BufferedImage into an int 2D array with 0 or 1:

```
int[][] A2D = imageToArrayRGB(figureAImage);
```

Every pixel in each row and column of the BufferedImage is iterated through, and converted to 0 (white) or 1 (black) based on its RGB value. For a grey pixel, a threshold was set so that it is considered as a black pixel if its value is above the threshold or otherwise a white pixel.

- **Methods implemented**

to find the transformation between figures:

```
private String getTransformation(int[][] a2d, int[][] b2d, int[][] c2d)
```

to compare two transformations by scoring differences:

```
private int calculateDiff(int[][] a, int[][] b)
```

to find the choices closest to expectation

```
private int getAnswer(int[][] expect, int[][][] choices)
```

- **Converting 3x3 to 2x2**

3x3 Raven problems are generally harder than 2x2, but sometimes, hard 3x3 problems can be easily solved when considered as a 2x2 problem.

Exemplary ways to convert 3x3 problems into 2x2 problems:

1) subsets of panels

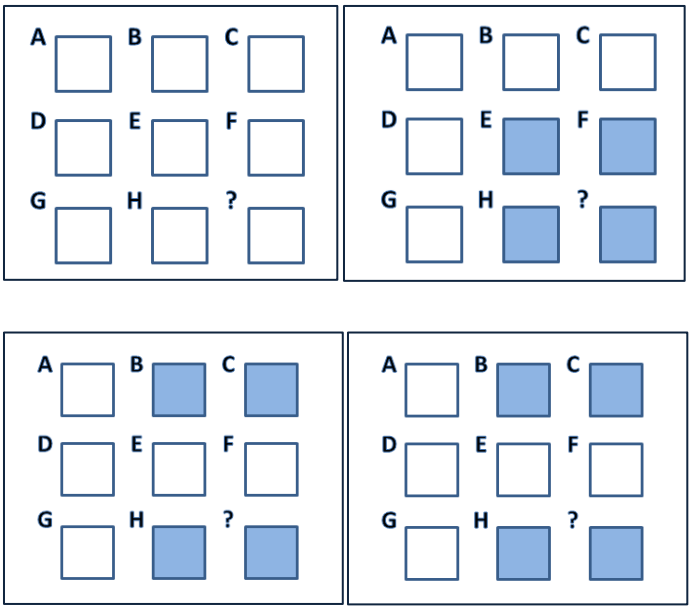

Figure 1

2) a single row/column

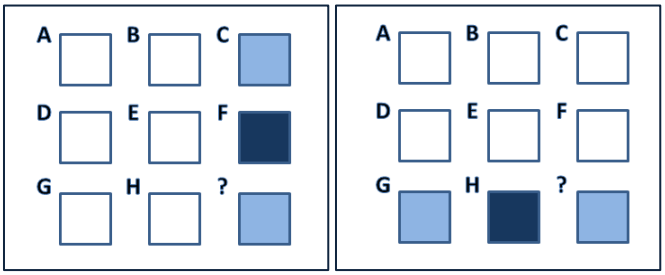

Figure 2

3) diagonal

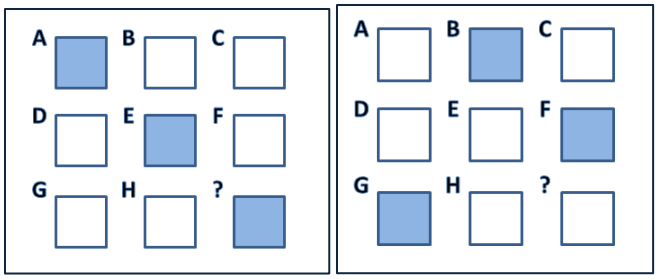

Figure 3

3. Result and analysis

Problem	Agent's Answer	Correct?	Correct Answer
Basic Problem D-01	3	Correct	3
Basic Problem D-02	1	Correct	1
Basic Problem D-03	3	Correct	3
Basic Problem D-04	1	Correct	1
Basic Problem D-05	7	Correct	7
Basic Problem D-06	1	Correct	1
Basic Problem D-07	1	Correct	1
Basic Problem D-08	4	Correct	4
Basic Problem D-09	3	Correct	3
Basic Problem D-10	1	Correct	1
Basic Problem D-11	3	Correct	3
Basic Problem D-12	3	Correct	3

Basic Problem E-01	1	Correct	1
Basic Problem E-02	7	Correct	7
Basic Problem E-03	2	Correct	2
Basic Problem E-04	8	Correct	8
Basic Problem E-05	5	Correct	5
Basic Problem E-06	8	Correct	8
Basic Problem E-07	3	Correct	3
Basic Problem E-08	1	Correct	1
Basic Problem E-09	7	Correct	7
Basic Problem E-10	8	Correct	8
Basic Problem E-11	5	Correct	5
Basic Problem E-12	6	Correct	6

Overall, I was able to solve all RPM problems in Basic Problem set D and E (24/24) within a few seconds. The results of Test Problem D and E are not bad either (9/12 and 6/12 correct, and one problem skipped) A few examples were chosen to illustrate how my agent solved these problems:

- **Identical**

For some simple RPMs, the figures in different panels are identical, as shown in Figure 4. By calling the method calculateDiff (A, B), my agent can compare the pixel-by-pixel the differences between pairs of Raven Figures. Ideally, calculateDiff (A, B) should return 0. However, sometimes these figures may have slight shifts that are easily ignored by naked eyes. Thus I set a small number as a threshold (for example, 100 pixels), and anything pairs of images with a smaller difference should be considered as identical.

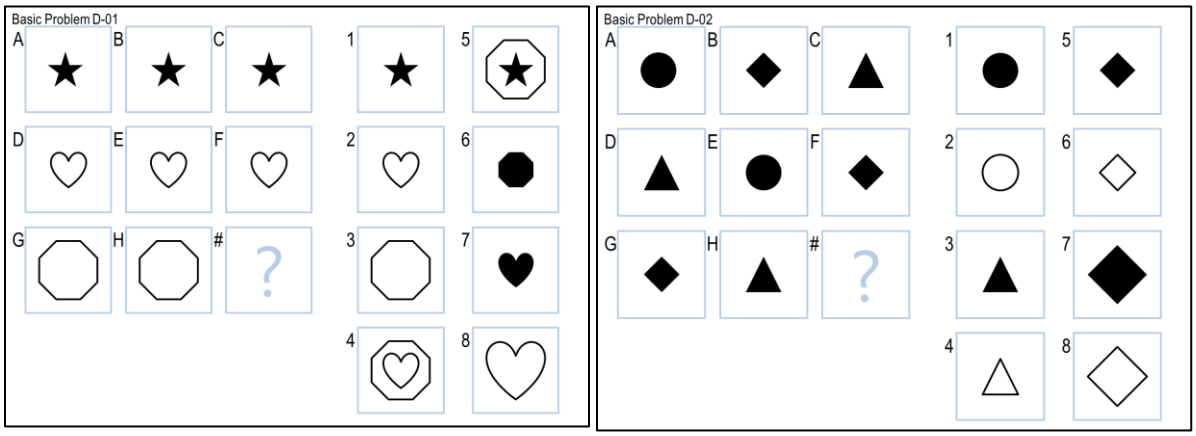

Figure 4 (identical examples)

Thus, if a certain problem satisfies the criteria (A is identical to B and B is identical to C), the transformation is labeled as "**identical ABC**" (Figure 4 left panel) or similarly "**identical ADG**" (vertical) or "**identical BFG**" (diagonal)(Figure 4 right panel). Then my agent will generate an answer based on an existing panel and pick up the best matching answers from choices 1-8.

❖ **Symmetry**

Some problems can be solved by symmetric transformations: vertical symmetry or horizontal symmetry. My agent defined this transformation to be "**mirror_leftRight**" (Figure 5, panel A to panel C). Similarly, horizontal symmetry can be detected (panel D to panel F) and defined as "**mirror_upDown**". So the agent will generate an answer based on existing panels, and compare it to choices 1-8.

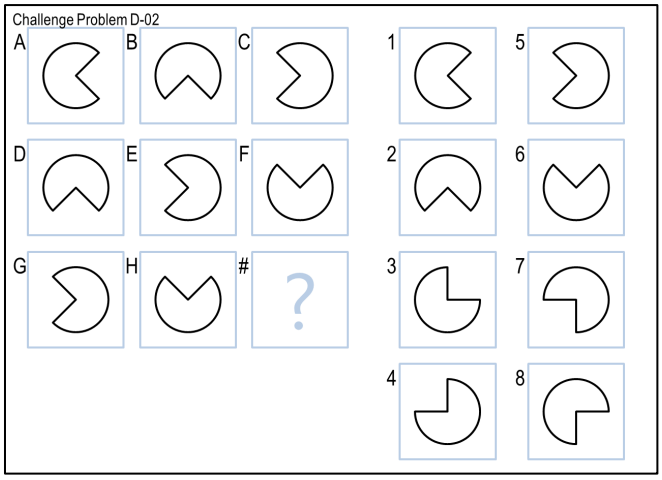

Figure 5 (symmetry example)

❖ **Set Operations**

A lot of problems can be solved by set operations: AND, OR, XOR and etc. For example, in **Figure 6**, overlapping of panel A and B generates panel C, and overlapping of panel D and E generates panel F. So my agent defined this transformation to be "**OR_ABC**". Overlapping of panel G and H will be the answer for this question. So the agent generates an answer based on panel G and H, and compares it to all the choices.

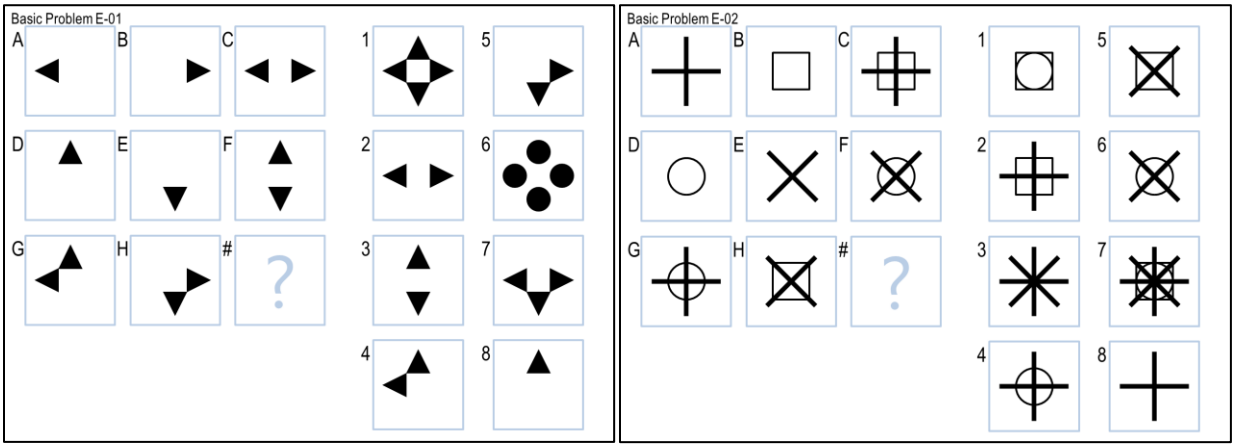

Figure 6 (OR examples)

Similarly, I can define "**AND_ABC**":

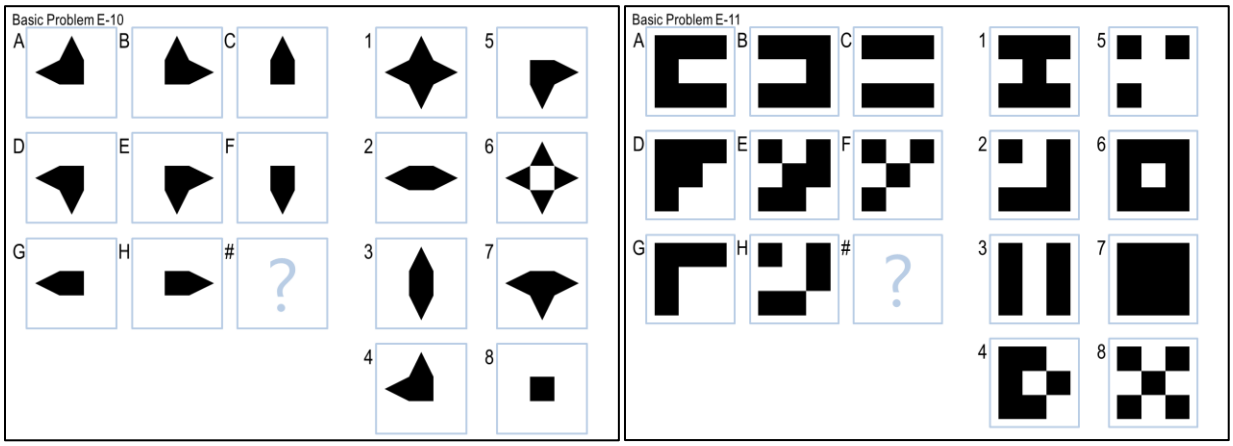

Figure 7 (AND examples)

"**Subtract_ABC**":

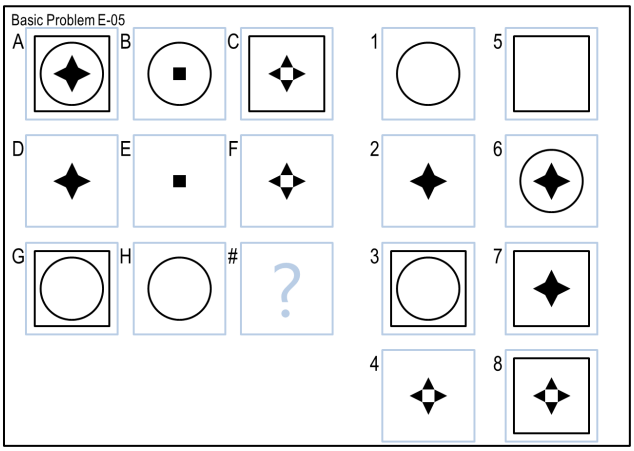

Figure 8 (Subtract example)

"XOR_ABC":

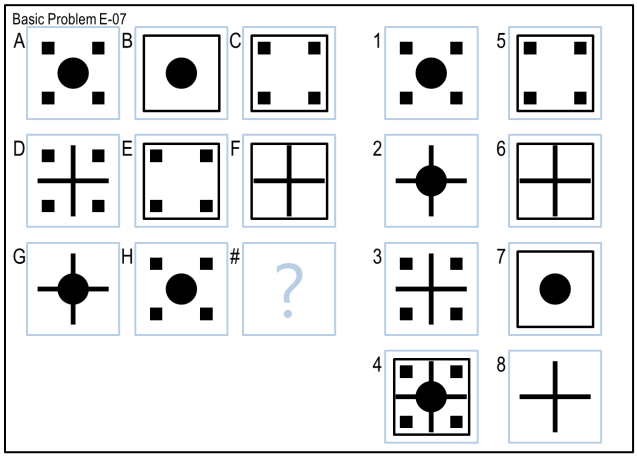

Figure 9 (XOR example)

Notice that in my agent, besides horizontal comparisons, all these transformations will be examined at vertical, diagonal and other possible combinations of panels.

- **Counting Pixel**

Although it is not quite intuitive in human cognition, an AI agent can solve quite a few RPM problems by just counting the number of pixels in each figure. An example is shown in figure 10, my agent found that pix(A) - pix(C) = pix(D) - pix(F), thus define this transformation to be "**increasePix_ABC**". Considering slight shifts of some figures, small differences are also accepted. Then it calculates the expected pixel numbers by pix(G) – (pix(A) - pix(C)), and then find the choice answer that has the closest number of pixels.

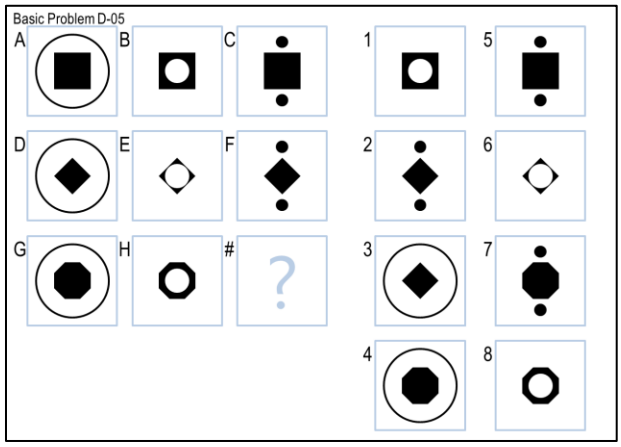

Figure 10 (pixel increase)

In some problems, the ratio rather than the actual number of pixels matters. For example, in Figure 11, pixels ratios in diagonal direction (A : F : H and E : G : C) are **3:4:5**. Then the pixel number expected can be calculated based panel B, panel D and the ratio of change.

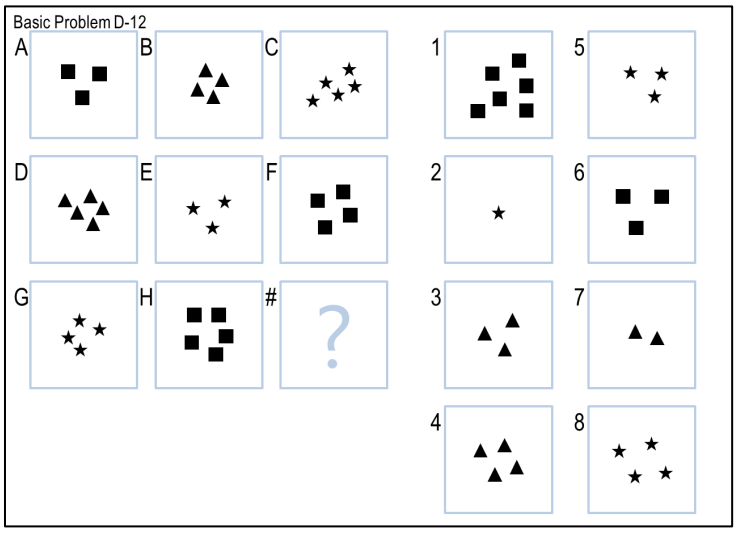

Figure 11 (pixel increase by ratio)

❖ **Problem remained**

This AI agent seems to be quite successful as it solved all RPM problems in Basic problem sets D and E. However, there are still many problems to be solved, which could eventually help to broaden and improve the current version of my agent. For example, some kind of problems, such as the one shown in figure 12, cannot be solved successfully at the moment, as my rather simple agent is not able to make correlations between open, shaded or filled objects of the same shape. I think that the ability to isolate and recognize individual objects will be critical for solving this kind of problems. A combination of visual and verbal methods may be the way to tackle these complex problems.

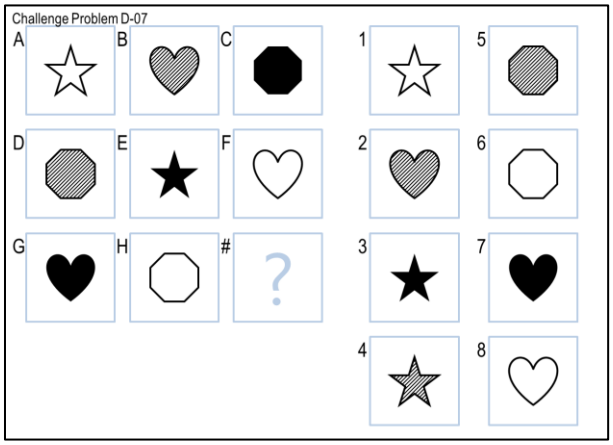

Figure 12

4. Discussion

Sometimes, a person can solve RPM problems with hidden rules that one does not even realize. For example, when we look at the answers, we pay attention to the most likely answer first. The computers can only iterate choices in order. Additionally, we constantly make logic deductions, generate hypotheses about correlations, then find violation to reject them/ evidence to confirm them. We gained these abilities most likely by receiving stimuli from the environment all the time. Our memory reinforces the experiences that we previously found stereotypes and helps us to better adapt to the nature. Thus the responses can be both helpful and illusive, but will not affect the reasoning of our computers.

For example, when we say two figures to be identical, for example panel A and B in figure 4 (left panel), we use definitions such as shape, size, fill, color, position. These are features that we use to examine the world. However, when AI read in these pictures as 2D array, it strictly compared data at each pixel, and will not be able to compromise unless intentionally designed to do so, and these panels will be considered different even though they looks the same for a real intelligence.

On the contrary, it is not very likely for human being to solve RPM problems in Figure 10 by counting pixels. One would naturally start to divide panels into objects and try to find correlations. However, it is so much easier for AI simply using pixel counting methods. Actually, counting pixels is such a straightforward and reliable method for AI and should always be given high priority in trying to solve RPM problems. These examples showed some of the most significant differences in reasoning by an artificial or a real intelligence.