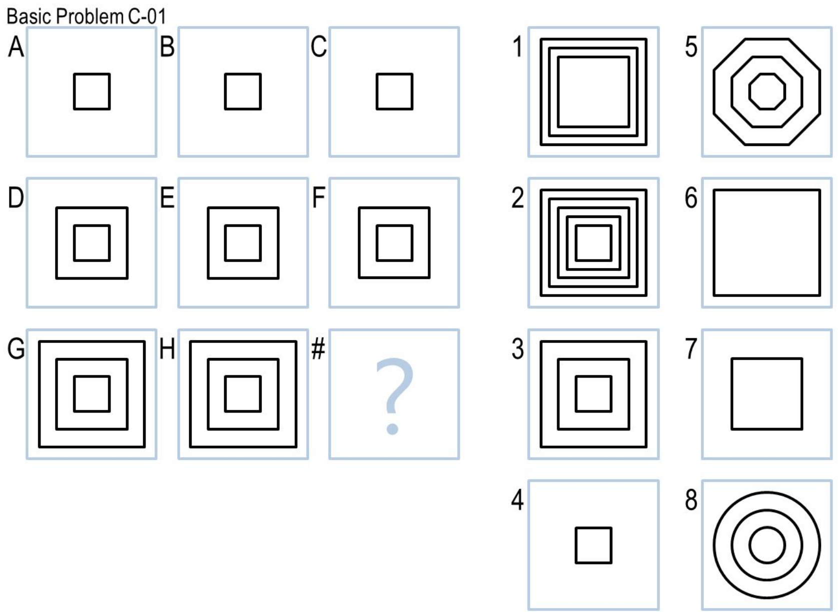
Traci Fairchild
CS7637 KBAI
Project 2
Project Reflection
October 25, 2015

Project Reflection for Solving 3x3 Raven's Progressive Matrices using Verbal and/or Visual Representations

Raven's Progressive Matrices

Raven's Progressive Matrices, referred to as RPM, is a group of nonverbal tests that are typically used to measure a person's reasoning skills. Reasoning is a skill commonly thought to be an adequate measure of a person's intelligence. The tests are a series of images in which one image relates to another in a certain way. The test taker is expected to analyze the relationship, in our example below, between the images below, looking for similarities or differences. The agent is expected to choose the image (images 1 through 8 below) that best completes the analogy of the first set. For example, a given 3x3 RPM test could look like the image below:



The Project

Project 1 was the first phase of writing and agent and Project 2 is the second phase of writing an agent that can pass this human intelligence test. The goal is to design the agent so that it uses any and all information provided to make a logical decision as to the correct answer for an RPM problem. This project is presented in this course as a way to expose the student to real world problems in the field of artificial intelligence, specifically, the cognitive approach. That is, how can artificial intelligence master the same types of thought processes that occur in human reasoning.

The Design

I chose to design my agent in Python (v2). For Project 2 I wrote a different set of heuristics to determine the problem's answer than I used with Project 1. This new logic performed very well with the 3x3 so I decided to rerun the 2x2 problem sets through it instead of the method I submitted for Project 1.

Upon instantiating the ***Agent*** class, and entering it's *solve()* function, the first thing my agent does is to create an instance of the ***AgentAnswer*** class. The ***AgentAnswer*** class is a new class for Project 2 and it creates a list of all the problem answer choices, which I will refer to as an *answer pool*. I then pass this pool of answers on to my individual solve methods. The intention is to examine each problem and, one by one remove potential candidates from the answer pool as they are determined they could not possibly be answers to our questions. Finally, when we are left with only one image in our answer pool, it is chosen and returned back from the agent.

So to begin, with the ***AgentAnswer*** class instantiated, I passed the problem and the answer pool over to a class that I developed for Project 1, ***AgentBasic*** (it's main procedure is called *solve()*). This class, in both Project 1 and Project 2, performs simple tests on the problem set. For Project 2, I needed to add more tests to this class for completeness in my ability to identify and eliminate answers that could not be possible. These simple tests were run one after the other, reducing the answer pool bit by bit, to just one remaining selection.

So, for each successive test within ***AgentBasic.****solve()***,** I would apply a simple test and based on the results of this test the agent would conclude that some answers in the pool could not possibly be accurate because they do not have a certain characteristic, as determined by the test. With this information, the ***AgentAnswer*** class was used to search through itself, that is, it's pool of answers and to remove from the pool those answers that did not apply.

Finally when only one remaining answer was left in our pool this selection was returned back to the Agent.solve() where, again, the answer would be inspected before moving on to the next stage of processing. The next stage of processing here was with the ***AgentSemanticNet*** class from Project 1. But my logic was so successful, however, that I did not have any use for the semantic net processing. It remains unchanged from Project 1 and so I will not discuss it's logic here.

My goal was to not rely on any one method for discovery, but instead to spread out the reasoning using as many tests as I could.

The Tests
Below is a description of the tests used to process each problem set. Every problem was run through these tests, each yielding its own unique results.

Equality Test 1
Looks at images A - H looking for identical images, such that A==B==C and D==E==F and G==H. In this situation the answer is determined to be equal to G & H. The answer pool is not necessary, the answer list is searched for the matching image and returned.

Equality Test 2
Images may appear equal visually but in fact they are not (off by a few pixels, etc). In this situation I examine the images more deeply with the function images_are_equal(). This function looks at the difference in the image channels and compares the results to a predetermined threshold. If the difference is below the threshold it is determined to be equal.

Blend Test
Determine if the images were just blended versions of each other. I had to abandon this test though, as it was not providing worthwhile results to examine.

Flip and Blend Test
Determine if the images were just flipped and blended versions of each other. Again, I had to abandon this test, as it was not providing worthwhile results to examine.

Same Number Of Objects Test
This test looks at the number of objects in each image. If the number of objects A==B==C and the number of objects in D==E==F and the number of objects in G==H then the answer must have the same number of objects as G&H. Therefore, I call a routine in the AgentAnswer class to remove all answers from our answer pool that does not have the same number of objects as G & H.

```
agtAns.removeNumObj_not(numObjsG)
```

Again, I performed the same test but with the outlying images only, A==B==C==D==F==G==H, excluding the center image and if test passed I eliminated from the answer pool any answer that did not have the same number of objects as A.

Lastly, the same test was performed using just the right and bottom images, C==F==G==H.

Pattern in the Number of Objects Test

The images might never have the same number of objects, but instead they may have a pattern to it. I came up with an algorithm that can detect a pattern in the number of objects as we move through our images. I noticed that the number of objects could be greater than, equal to, less than or multiples thereof, the number of objects in the image preceding. I used the addition, multiplication and subtraction operators to determine a pattern. With these operators I then looped through 2 variables, x and y, both in the range of 0 - 5, looking for a pattern:

```
for op in ('+','*','-'):
  for x in range(0,5):
    for y in range(0,5):
      op_func = op_functions[op]  # ex: + is defined as operator.add
      numObjsB == op_func(numObjsA,x) and\
      numObjsC == op_func(numObjsA,(x+y)) and\
      numObjsE == op_func(numObjsD,x) and\
      numObjsF == op_func(numObjsD,(x+y)) and\
      numObjsH == op_func(numObjsG,x):
```

When this statement evaluated to true that indicated a pattern was found and the answer must have x <some operator> number of objects than G and y <some operator> number of objects than H. I used the same logic to determine the final number of objects the answer must have, and removed all others as described above:

```
agtAns.removeNumObj_not(op_func(numObjsG,x+y))
```

Again, I performed the same test but with the outlying images only, **C**==**F**, **G**==**H**

Pattern in the Number of Fills Test

I performed the exact same test as the *Pattern in the Number of Objects Test* but instead of looking at the number of objects, I looked at the number of fills that a problem had.

Image Uniqueness Test

For each image in our problem set, if it is NOT repeated anywhere else in the problem set then it is NOT likely to ever be a correct answer. If it is found in our answer pool then it should be removed. I used a function called elim_image() to look at each image in the problem set, if it is not repeated then remove it from the answer pool if it is present:

```
for x in ('A','B','C','D','E','F','G','H'):
  # if none of the images match image x, then image x is likely not an answer
```

```
if self.elim_image(hasVerbal, problem, x):
    agtAns.removeImage(problem.figures[x].visualFilename)
```

Figures Common To All Objects Test

I created a function which will loop through each image and return a list of objects that are common to all the images, such as [‘square’, ‘circle’]. Then I called another function from my AgentAnswer class to remove any answers from our answer pool that does not have these two objects.

```
shape_list = self.find_common_shapes(problem)
agtAns.removeShapeObj_not(shape_list)
```

Crop Test

In this test I cropped a portion of each image and compared to the same cropped portion of the next image. If each image had the same cropped portion then the answer must also have the same cropped portion.

```
rc, croppedImg = self.crop_and_test(problem, 'A','B','C', 'bottom-third')
if rc == True:
  rc, croppedImg = self.crop_and_test(problem, 'D','E','F', 'bottom-third')
  if rc == True:
    rc, croppedImg = self.crop_and_test(problem, 'G','H','H', 'bottom-third')
    if rc == True:
        #answer must have same cropped portion, eliminate those that do not
        agtAns.removeCropped_Not(problem, croppedImg, 'bottom-third')
```

I performed this test two times, once on the bottom third of the image and again on the bottom two thirds of the image.

Number of Answer Left Test

At this stage I have performed a sufficient number of tests and hope to have only 1 answer left in our answer pool. If so, then this must be our answer:

```
if agtAns.get_num_answers() == 1:
   answer = agtAns.get_fig_num(0)
```

If not then I perform a few more tests on equality, but use a higher threshold. I did not do this earlier with my other equality tests because I noticed it was giving me a lot of false positives. So I thought it better to get the solid tests done first rather than expose the process to more faulty observations.

Pattern in the Number of NonFills Test

This test tried to determine if the number of non-fills was increasing, decreasing or staying the same. This test is different from the *Pattern in the Number of Fills Test* in that this test does not seek to find an exact number in the pattern as above, rather it looks to find whether the number is increasing, decreasing or staying the same. This test is rather narrow that is why it is performed last, if no answer has yet been found.

```
#get the number of non fills and calculate the differences
incremental_amt1 = numNonFillsC - numNonFillsF
incremental_amt2 = numNonFillsG - numNonFillsH
if incremental_amt1 == incremental_amt2 and numNonFillsF == numNonFillsH:
    #we have a pattern
    if incremental_amt1 < 0:  #means the number is increasing
       num_nonfills = numNonFillsF + abs(incremental_amt1)
    elif incremental_amt1 > 0: #means the number is decreasing
       num_nonfills = numNonFillsF - abs(incremental_amt1)
    else:
       num_nonfills = numNonFillsF

    agtAns.removeNonFillNum_not(num_nonfills, '=')
```

The Classes
Below is a summary of the classes which my agent uses:

Class Name	Source File	Description
Agent	Agent.py	The main agent class
AgentBasic	AgentBasic.py	The class that performs all the tests described in this document
AgentAnswer	AgentAnswer.py	loads all problem answers into a list and defines functions related to this list, such as show_prob_answer_list() or eliminate_answer()
Attribute	AgentAttribute.py	class for a single attribute, defining variables and functions for testing and processing an attribute
AgentImage	AgentImage.py	helper class used for performing common functions on input images
AgentSemanticNet	AgentSemanticNet.py	the class used to perform the semantic network logic. Not effective in Project 2.

Project Reflection
The following questions were presented as part of the project to enable the student to more closely reflect upon the design choices made and the outcomes of the project.

- 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?

In Project 2 I used the process of elimination, performing a series of tests one after the other, hoping to reduce the set of answers down to just 1 possible solution - our answer. I had this idea in Project 1 but I was uncertain of it at the time and thought of it a little too late so I "tabled" it for Project 2.

I'm not sure I would call this method risky, however, I am curious as to how many other students went in this direction. Risky or not, it did pay off. Each problem set had a few reliable characteristics to work with, such as number of objects, number of fills, which I developed tests for the possible combinations of these characteristics and the results were quite good.

- 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 agent performs test after test in hopes of reducing the number of allowable answers. The answer is selected when only 1 remaining possibility is left.

The most important metrics used in my tests are the number of objects that a figure has and the number of fills/non-fills that a figure has. With this information I was able to examine the images in several different ways, looking to identify patterns. Another metric used periodically in the tests was a variable threshold value in image comparisons.

This is an exact correct answer, there is no rating, nor scaling to choose from. If the number of allowable answers is not reduced to 1 then the tests continue. If I reach the end of my problem solving and there is more than 1 answer to choose from then I skip it.

- 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 does not have logic in place on what to do if more than one answer remains. I did not want to make a guess because we get penalized for guessing so I just skip the problem. I'm not sure how I could logically assign weights to any answers that remain. To me this doesn't make sense so I did not implement it. This could be a fundamental problem with my design. I'd have to come up with tests that could examine the remaining images, rather than examining the problem set images. I'm not sure how I would approach this, what tests or metrics I could seek to discover that would add more clarity on the correct choice. Given enough time and examples I could solve this limitation but for now I am left skipping the problem.

- 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?

I need to find a way to process the images without the use of the verbal representation. While I managed to land on some ideas that work for Project 1 and Project 2, I still have a long way to go for P3. I find working with the verbal representations frustrating and it feels like a cheat. But I need to find a way to identify all the metrics I use in my existing tests - # of objects, # of fills, etc. I think I will do this by selecting all the single images in all the problem sets and use them to see if they exist in my problem set. If they do, I can use some of the verbal representation.

Throwing out the reliance on the verbal representation would be a big step in designing an agent to think like a human. I am not sure I will be able to overcome this, because I don't have much experience in manipulating and analyzing images graphically but I will try. I suspect I will still need to use some verbal representation if it is available to me.

I think this improvement would maintain my agents accuracy, yet falter on its efficiency as it would take longer to run. Generally speaking, this improvement would make my agent closer to thinking like a human because it would not have to rely on any verbal representations.

- 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?

I feel my agent performs efficiently, though not instantaneously. For Project 1 it took almost a minute to perform 24 problems, now it takes approximately 30 seconds to process the 48 problems in Project 2.

Project 1 (24 problems): Approximately 50 seconds
Project 2 (48 problems): Approximately 30 seconds

My accuracy for Project 2 is fair. I have not made big strides with the challenge problems but the basic problem results were acceptable:

Set	Correct	Incorrect	Skipped
Basic Problems B	12	0	0
Challenge Probs B	3	3	6
Basic Problems C	11	0	1
Challenge Probs C	1	11	0

Some other metrics I could mention are process metrics such as human effort used, time spent on the project, conformity to the project requirements, etc. I used a lot of time designing tests.

- 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?

Some of the equality tests allowed me to forgo the verbal representations but my code relied heavily on the verbal representation. Luckily, I did not rely on all of the characteristics of the verbal representation. For example, I had no need for the following attributes: size, width, length, angle, inside, left-of, above, and overlaps. The only attributes I needed were the shape and fill. I still need to find a way to identify the shape and the number of figures in an object for Project 3. Possibly more depending on the rigor of the problems.

- 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?

For Project 1, I was feeling rather dubious about my process and that it never even came close to thinking like a human. To me it was just another coding problem that I had to account for all the scenarios I could. I was feeling unimpressed with the cognitive approach to AI if this was all it was. But, in Project 2 I feel like I managed to move away from this mindset, if just a little bit. I felt my results approached the realm of AI, if only within a fraction.

By performing test after test, I felt at ease with the coding process. Breaking it up into smaller chunks is, again, what humans do naturally.

By eliminating the answers as I did, I feel like this is definitely the way a human would look at it - one by one deciding what should not belong. I think that humans do this almost constantly, without knowing it - always aware of what is normal and what is not. If something is not normal, then it indicates a problem of sorts to be solved.

I believe these methods come naturally to humans. As a survival measure, it keeps us in control of our surroundings and in our comfort zone.