Project 1 Reflection

Introduction

The path that I took in building an agent capable of solving the *Raven's Progressive matrix problems is to*[1] implement an intelligent generator, followed by an intelligent tester. There are several challenges that arose early on during the implementation process which will be discussed below; translating abstract theoretical ideas into practical applications is always a challenging task. Part of this challenge is due to the fact that limitations arise in terms of efficiency and data management.

Approach

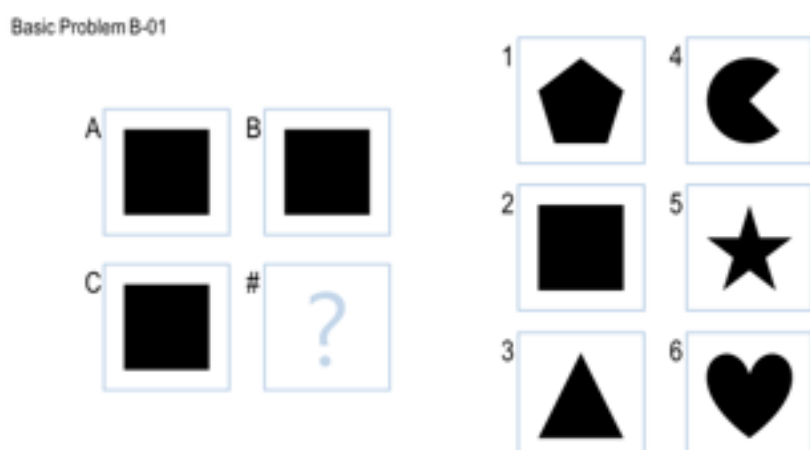
One of the first thing that I had to deal with right at the beginning was to determine how to manage the problem data. For example, The problem, as described in the *ProblemData.txt,* maintains a certain format. Therefore, several assumptions had to be made about the structure of these problem descriptions. Some of the assumptions that I made were as follows:

- Every single object described is unique
- There exist a set of known *reference object properties* (i.e "inside", "outside", "above")
- There exist a set of known alignment properties[2] (i.e "top-left", "bottom-left", etc)
- Objects that are rotated, are always rotated a common axis[3]

Making these assumptions, I have learned, yielded a fairly positive outcome for the agent.

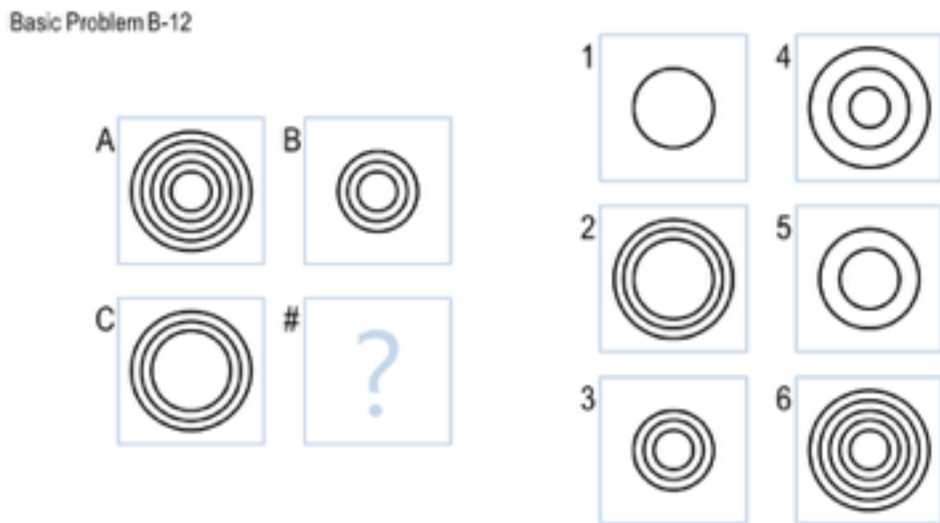
Generator

As stated earlier, I have implemented an intelligent generator, in an attempt to generate the answer. In a very basic case, as in the case of *Basic Problem B-01,* The generator is very straight forward: Object a, maps to object b—therefore, the transformation that occurs between the two object is the same transformation that should occur between *C* and the answer.



[1] The Agent is currently able to generally solve 2x1 and 2x2 problems correctly
[2] Object alignments are used to detect reflections
[3] Making this assumption allows for possible reflection detection due to rotations. This only works, however, if the axis is consistent between the different figures.

In order to challenge the generator, however, lets take a look at a more complicated problem, like *Basic Problem B-12*:



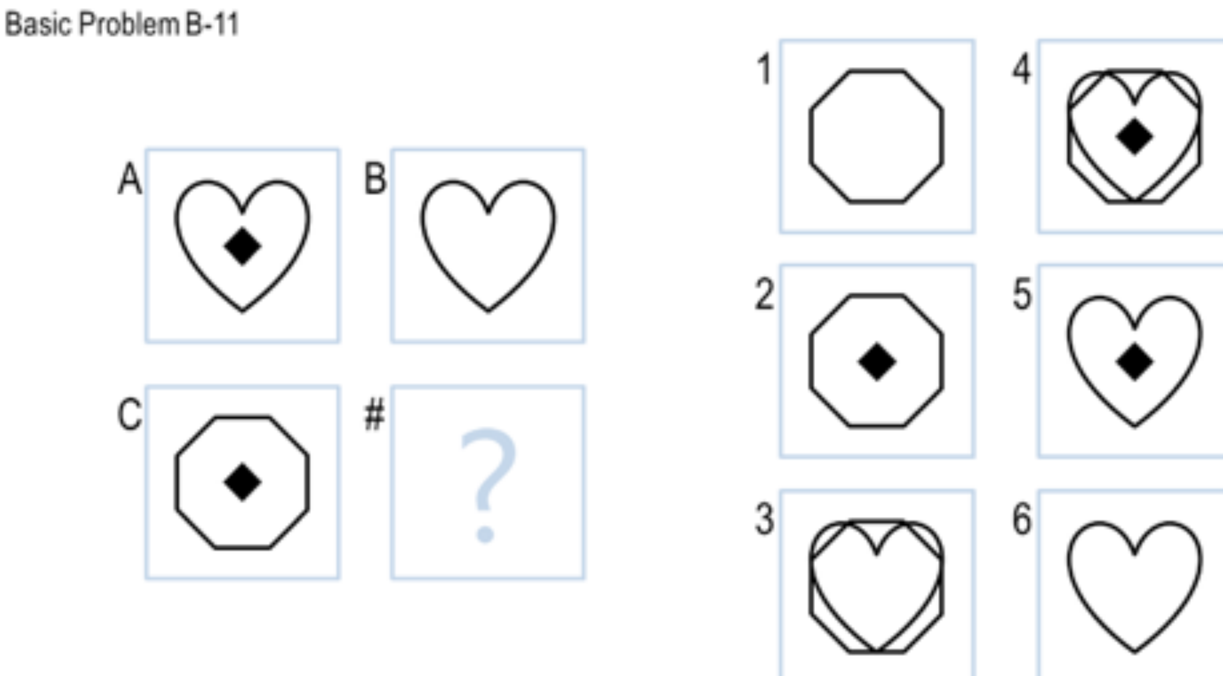
In this problem, an obvious observation is that all of the objects are circles. The challenge that arises to the generator, therefore, is which circle in **A** map to which circles in **B**. This mapping questions is not hard for a human to answer visually, but for my generator, extra steps are required.

Any time multiple objects are found in a figure, the generator first calculates the difference in number of objects between **A** and **B**. In this case, we notice that Figure **A** has 5 objects, while Figure **B** has 3. Therefore, the difference between the number of shapes between **A** and **B** is **2**. Without further computation, the generator can already eliminate the answers from the answer set based on this simple fact: whatever transformation were to take place from **C**, the end result should differ by exactly 2 objects[4]. In this particular problem, every single answer is filtered out except from answer **1**. The following table illustrates how this filtering process takes place:

Figure	Objects	Difference
1	1	2
2	3	0
3	3	0
4	3	0
5	2	1
6	5	-2

[4] When speaking of differences here, I mean in terms of existence (i.e whether the object exists or not) — The number of objects in Figure C should differ by the same amount of objects that Figure A differed from Figure B.

Therefore, without further computation, the correct answer has been filtered out the generator having to generate any transformations. In cases, however, where there does not exist such obvious differences in figures, the generator continues by generating all possible object mappings. Consider this example from problem



After the initial filtering by the generator, the agent reduced the possible answers to either **1** or **6**. At this point, the generator must continue execution and determine how Figure **A** transforms into Figure **B**, and apply that transformation to **C** in order to choose the correct answer. To start, however, the generator must determine which which object in Figure **A** maps to which object in Figure **B**. For the human, this is very trivial task: it is obvious that the heart in **A** maps to the heart in **B.** The generator, however, is unaware of the shapes that are involved. It makes no distinction between the heart and the diamond, therefore, the only option is to assume that both possibilities are equally probable. As such the generator generates a permutation of all possible combinations between the objects in **A**, and the objects in **B**:

```
▼ object_permutations = {list} [[('a', 'DELO'), ('b', 'c')], [('b', 'DELO'), ('a', 'c')]]
      __len__ = {int} 2
   ▶ 0 = {list} [('a', 'DELO'), ('b', 'c')]
   ▶ 1 = {list} [('b', 'DELO'), ('a', 'c')]
```

As observed above, there are two possible scenarios:

Object	Transformation
a	Deleted
b	C

OR

Object	Transformation
a	Deleted
b	C

Each possible scenario is observed by the generator, however, an important step that the generator performs, is it determines which transformation is quickest[5]. In this particular case, the quickest transformation From Figure **A** to Figure **B**, is to delete the *diamond* from Figure **A**. The generator, at this point applies this lowest transformation to Figure **C**, and passes the control over to the Agent Tester.

Tester

Although the generator does much of the heavy lifting in this intelligent Agent implementation, The tester does, however, add another layer of intelligent to the whole program. For example, when a list of answers are generated, the ones which are not even part of the answer set are automatically discarded[6]. In the end, a list of candidate answers are left to choose from. For typical problems, the candidate answers list is usually very low. It is however, problematic when the tester is not capable of reducing the answer set all the way to 1 (i.e. the correct answer), or worse, reduce the problem set to 1 but to the wrong answer. A way to improve on the probability of arriving to the correct answer, I have learned that there can exist multiple transformations that result in a particular answer because of the nature in which the generator generates the transformations[7]. As such, I maintain a list of possible answers in a priority queue, and every time an transformation yields a particular answer, its priority increases. As a result, when selecting the best possible answer, the tester always selects the answer which has the highest priority in the answer queue.

For all of the given input problems in the Basic Set, the Tester was able to successfully select the correct answer from the priority queue data structure.

Mistakes and Challenges

As mentioned earlier, the Agent makes assumptions about the referenced object properties. For example, objects are not distinguished when they are "inside", or "above" other objects. This could result in false positives when determining whether two figures are in reality identical. This problem is further complicated due the fact that all permutations of a transformation can exist in a given problem.

To remedy this issue, a data structure must be implemented to maintain reference mapping between these special object attributes. Implementing a mechanism to maintain the references is challenging on many levels — for example, when considering property "inside: a,b", the swapping of these references must also be considered, "inside: b,a".

Other mistakes that my Agent makes are those that involve objects which have angles as properties. The agent is aware of how to interoperate angles[8], as such, it assumes a few select angles when encountering these properties. Furthermore, the Agent is completely

---

[5] A "quick" transformation is one which requires the fewest amount of mutations.
[6] Note that at this stage in the program, The answer may be already reduced due to the workings of the generator prior to the tester taking control.
[7] Permutation of all possible combinations of object combinations.
[8] A set of angles are hardcoded in order to detect reflection due to rotation of certain objects.

unaware of shapes in figures. Because of this, the generator creates a list of all possible permutations. This can be very inefficient for figures which contain many objects[9].

Conclusion

Looking back, I think the Agent is fairly limited primary, however, by the way that the Agent is working with the information available. Some of the biggest challenges had been to determine how to process the *Verbal* data that problem provides. The Agent had to be made aware of this data structure and the special case properties mentioned earlier[10]. For better performance and generalization, its better to load the image directly using vision frameworks like OpenCV. This would enable the agent to read the image in a generic form agnostic toward the properties of objects in figures. Doing this, would certainly make the agent more efficient[11]. Lastly, a CV approach more closely mimics human strategies of solving these kinds of problems. For example, in CV the agent would not be concerned of how a particular object is labeled[12], but would just read that object in as data. Similarly, the human mind doesn't care to have reference to an object, we just visually are able to reference it. Using the verbal representation of the problem data, my agent didn't quite solve problems similar to how humans solve such problems due to its combinatorial nature and its trial and error approach. Furthermore, the agent does very little initial observations about a given problem. There is no concern of obvious patterns in the input image. Humans, however, have a distinct ability to look at a given problem and almost immediately be able to observe certain patterns and features which would otherwise go unnoticed by a KBAI Agent.

---

[9] Basic Problem B-12 has many shapes in a single figure. The generator generates 120 possible permutations of possible mappings between the objects in Figure **A** and the objects in Figure **B**.

[10] Reference properties like "inside", "above", etc

[11] There is a level of redundancy in how the agent currently calculates similarity. A CV approach for this would be a significant improvement.

[12] All objects in the Verbal representation data which was used by the agent had labels indicate how objects relate to each other.