CS 4635 - Knowledge Based AI
Brad Ware
Project 2 Design Report

Contents

1. How does it work?

I took the approach that my agent had last time and tried to build on that foundation. In Project 1 I implemented an agent that tried to find the most similar differences between figures and then returned the one with the best result. I build on that algorithm this time with a more sophisticated mapping that would allow more elimination of possible answer choices.

A. Strategy

From a very high-level perspective, the strategy is to analyze the differences between figures A and B, and then choose an answer choice that reflected the most similar differences when compared to figure C. Then, that result would be stored and the exact same strategy would compare figures A and C and choose an answer choice that reflected the most similar differences when compared to figure B. By using this strategy, I would not just find relationships between two figures, but would find the **key** characteristics that would determine which figure would be the right match for C. However in my last implementation, many of the answer choices were giving the exact same score based on my system, and I really didn't have a sophisticated way to break ties. This time I used the preliminary mapping to weed out answer choices that are easily eliminated and then analyzed further based on key foundational differences.

The agent stores figures A and B in a separate wrapper object that gives a score to each figure's object based on its differences in many categories. The same is applied to figures A and C. It then stores each object index in a hash table that indexes to another hash table storing the attribute category and transformation that occurred.

This same step is repeated for each answer choice and is then stored in a separate wrapper object with figures C and B.

After that, comparisons are made between the hash maps of C and an answer choice, and the one's of figures A and B. This is also repeated for B and an answer choice compared back to the hash maps of figures A and C. If the score is the same for an object, then the answer choice's score is incremented. Each level of similarity increases the score more, that way clearly wrong answer choices are eventually eliminated much faster. The figure with the highest score is then returned as the answer.

However, because ties can occur frequently, a list is used to keep track of all the answers that tied with the largest score.

If the list is greater than 1, the real elimination heuristics occur that compare many categories trying to find differences that can differentiate answer choices from another.

Finally, the answer choice with the highest score based on it's similarity differential rating back to the Pair's of figures (A,B) and (A,C) is returned as the answer.

B. Architecture

Three helper classes were defined so that one could easily compare and contain the information about each figure and it's relation to another.

Class Pair (Ravens Figure, Ravens Figure)

The purpose of this class was to track the relationship differences between each of the objects in the different figures. This class kept a hash map which stored numerical indexes for each Object and that index hashed to an attribute in a different table. These attributes stored a difference string trying to describe what transformation occurred from figure to figure.

Variables

- *fig1*: the first RavensFigure to be compared.
- *fig2*: the second RavensFigure to be compared.
- *score*: the score of the relationship between the two figures.
- *scoreMap*: a hash map containing each object index as the key and another hash map storing the associated transformation with the attribute.
- *diffNumObjects:* string value that compares the different number of Objects that each figure contains and returns the resulted transformation.

Key Functions

- *setUpScoreMap: initializes the hash map with the correct object index and builds the helper map that goes into more detail about each attribute change.*
- *getfDiffNumObjects: returns the transformation from the different number of objects.*
- *getfFillDifference: returns the transformation from the fill difference of objects.*
- *getSizeDifference: returns the transformation from the size difference of objects.*

Class ObjectPair (Ravens Object, Ravens Object)

The purpose of this class was to track the relationship differences between each object in a figure. This class kept a hash map of characteristics and a **string transformation** keeping track of the differences. Any different characteristic was stored in the hash map, and it was taken into consideration the number of characteristics and the differing possible values.

Variables

obj1: the first object to be compared.
obj2: the second object to be compared.
attrMap1: a hash map containing all of the attributes of obj1.
attrMap2: a hash map containing all of the attributes of obj2.
diffMap: a hash map containing all of the associated differences in attributes of the two objects.

Key Functions

compare: made an ArrayList displaying all of the attribute categories that were different between the objects.
setUpDiffMap: used the difference list to build a HashMap storing the category and transformation.
angleDiff: used to track the numerical differences between angles.
sameShape: determined if the object contained the same shape and stored that boolean value in the hash map.
sameSize/Fill: determined if the object contained the same size/fill and stored that boolean value in the hash map.
getSame… : getters for each of the boolean attributes comparisons
getValues: getter for the hash map containing all of the attributes and their boolean values

Class AttributePair (Ravens Attribute, Ravens Attribute)

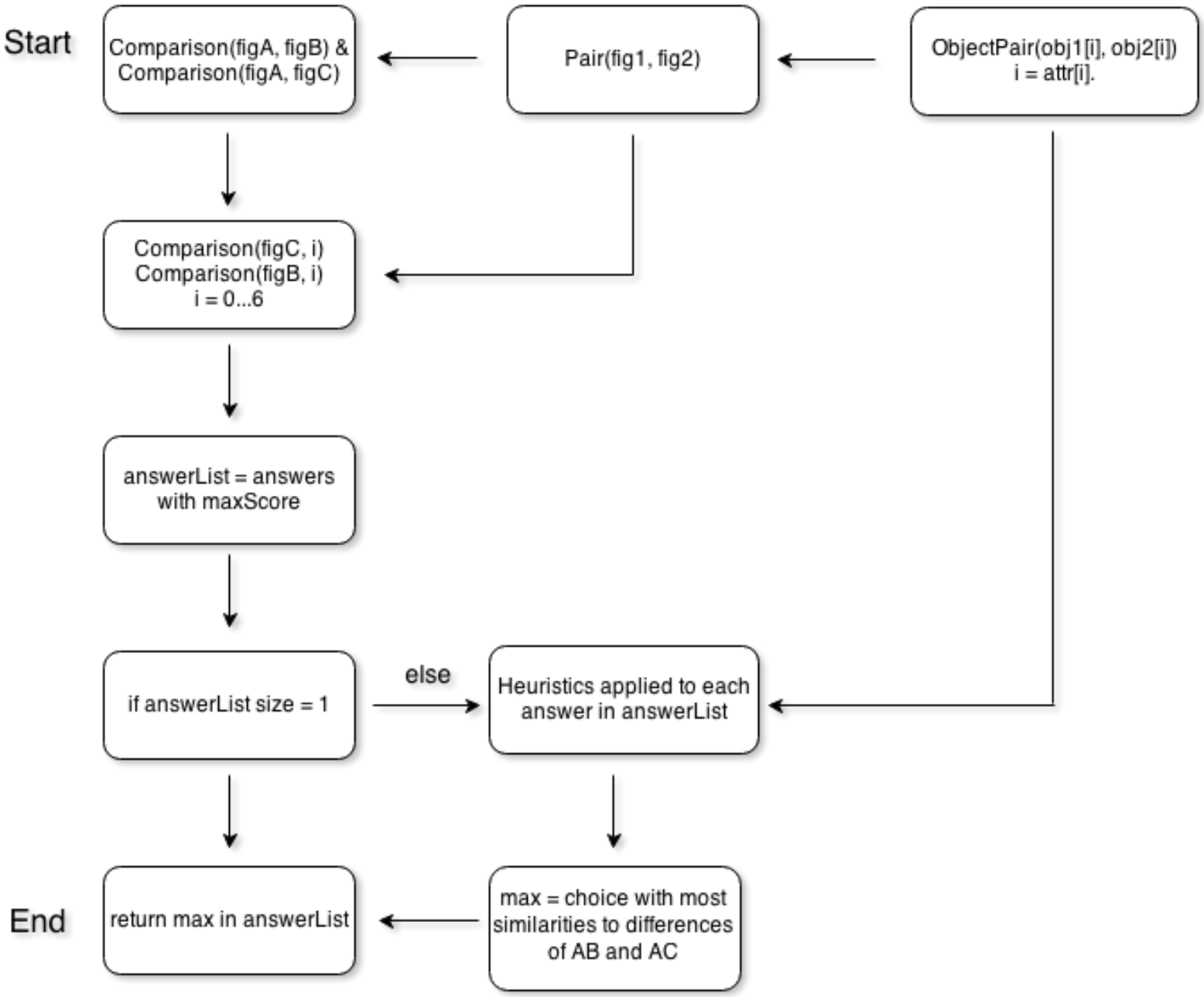
The purpose of this class was to track the relationship differences between each Ravens Attribute in a figure. Simply stored the transformation difference between the two attributes that were of the same category.

Variables

attr1: the first attribute to be compared.
attr2: the second attribute to be compared.
index: the attribute's name that was used to store the category.
diffValue: a string that stored the difference between each attribute.

Key Functions

getDiffValue: returned the transformation between each attribute's value.
getIndex: returned the index/name that was categorizing this attribute.

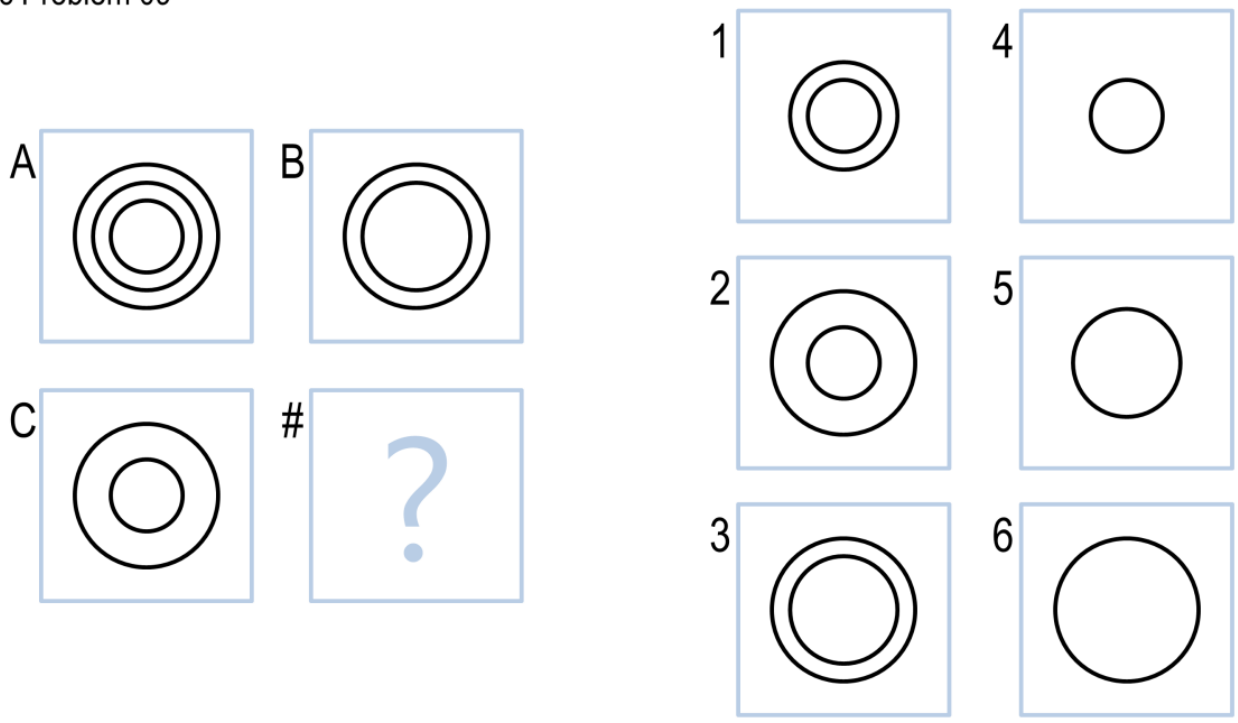


Flow Diagram of how Solve finds Answer

2. Application

We will walk through an example using the strategy I implemented to show how the agent solves a Raven's progressive matrices problem.

2x2 Basic Problem 09



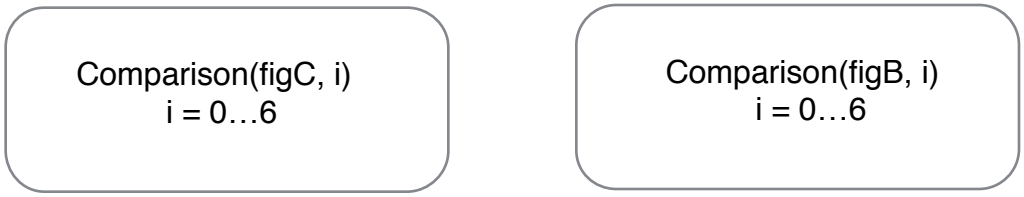
A. Example Problem Walk-Through

Comparison(figA, figB)

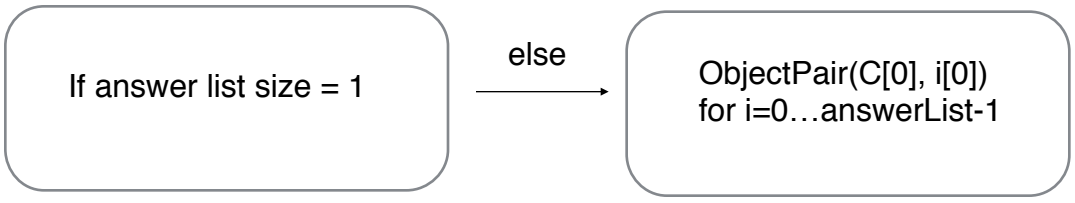
Comparison(figA, figC)

Even though figures A B, and C have multiple shapes, there are some discrepancies that can be used to eliminate some of the initial answer choices. Here is an example of what the two hash maps might read:

Hash Map of Figures A and B & HashMap of Figures A and C	
Shape	SAME
Size	VARIES
Fill	FALSE
Angle	SAME
Inside	REMOVED
NumShapes	REMOVED



Therefore, when we run the comparison on all of the answer choices to B and C, notice that it needs to contain one less object of the same shape that was removed from the inside.Therefore if figure C has one object inside of it, then our answer can't have any objects inside of it. We then can eliminate answer choices 1, 2, and 3.



Because our answer list will contain 3 values, we will iterate through each choice and apply different heuristic functions on it and the more it passes the higher the score it

gets. These heuristics weed out the other answer choices that appeared to be correct, but weren't after further investigation.

Scores of Figure 2, 5, and 6	
Figure 4	10
Figure 5	10
Figure 6	12

The main difference is the heuristic function that checks to see the size difference of all of the objects. In transformation of figures A to B and A to C, the smallest object in the first figure is removed from the second. However, the largest object is kept. Therefore, when analyzing figure 4, 5, and 6 and comparing them to figure B and C, we need to at least keep the largest object. Figure 4 and 5 keep the smallest and middle object in relation to size, but only figure 6 truly keeps the largest in both of the transformations. That's why figure 6 score is 2 points higher than the other two, because it received an extra point for both B to 6 and C to 6.

return answer = "6"

3. Shortcomings

Obviously, this agent does have some downfalls. One of the biggest issues is determining not just when an angle change occurs, but what exactly this change equates to in transformations. Also, being able to determine the the exact depth of change for multi-shape objects, and applying those changes

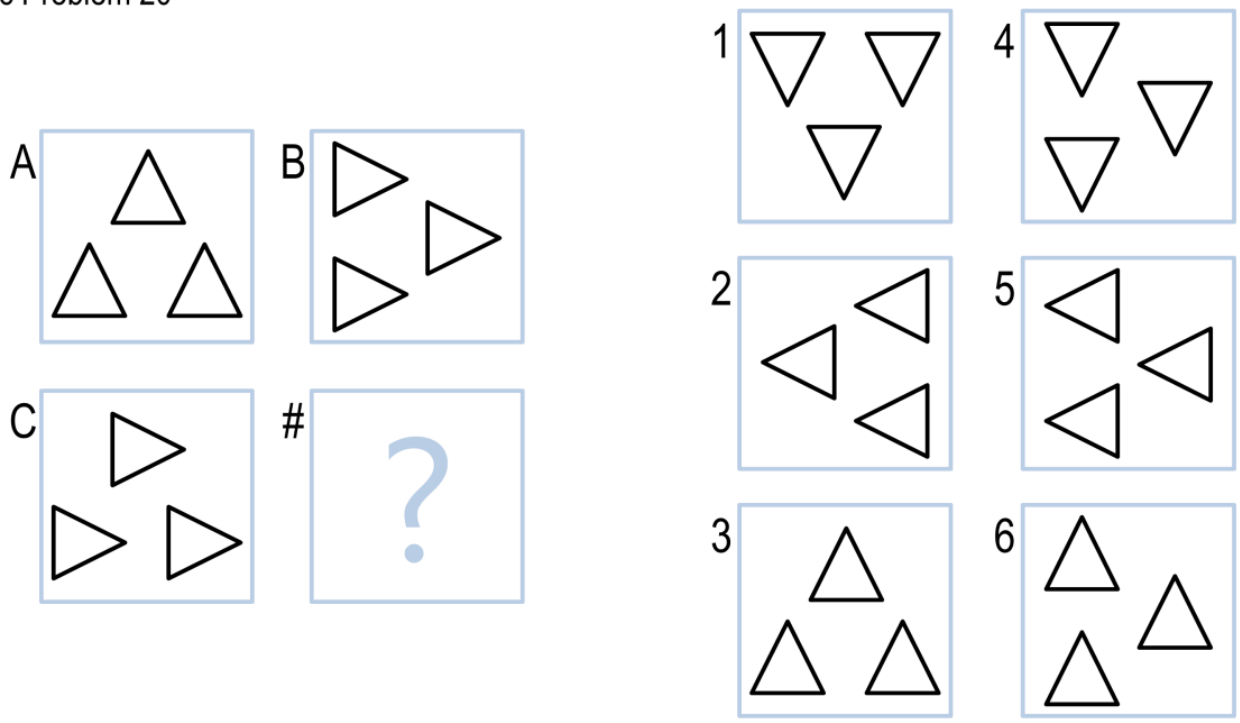
A. When does it fall short?

My agent usually fails when there are multiples object involved in the change, and when the angle determines a pattern. For the human eye it's very easy to see a pattern for how the shapes should rotate, however when designing an agent the math can become extremely tricky, and hard to generalize across multiple scenarios.

B. Why?

I thought it would be easiest to demonstrate this through a short example.

2x2 Basic Problem 20



Right now, my agent will store figures 1, 2, 4, 5, and 6 as possible answers in my answer list. Figure 3 is eliminated because it is an exact replica of figure A, where only in very unusual circumstances will a repeated image be the answer. However, it can't distinguish between slight variations of when shapes are to the "left-of" each other, and when that left is necessary. It also has trouble with picking up patterns of "rotating down" or "rotating up" as it just knows a numerical value to turn but not the direction. This becomes hard to solve when there are multiple answer choices that are rotated the same degrees but just in different directions.

4. Potential for improvements

A. What could have been done better?

The agent could have been much better organized. Instead of just incrementing the score value based on a few key heuristics, it could have labeled the type of problem based on certain characteristics such as are angle changes involved? Do the number of objects change in each figure? These sort of generic questions could break the problem

down into certain sub-areas that would make it much easier to solve. This could be accomplished by the checkAnswer function, which would have been a great way to analyze not only the type of problem but also the answer associated with it.

Another way my agent could have improved is breaking down the angle changes. It's one thing to numerically know when an angle changes, but a totally different spectrum to understand how that change replicates a transformation. Sometimes my agent had difficulty distinguishing the difference between a rotation of 90 degrees to the left, and 90 degrees to the right (actually in all directions) which would have been very helpful when trying to eliminate answers.

5. Run-Time Analysis

A. How can it be solved more efficiently?

I have yet to see any real efficiency gaps because of my implementation, but that's mainly because the problem space is so small. For instance when AI agents are trying to play chess, the branching factor can become huge, making efficiency one of the most important things. To analyze how my agent runs the solve function, I am going to break down each of the steps it runs through, and try to quantify the run-time there:

Iterate through each answer choice (figure): *O(n)*

Iterate through each object of each figure: *O(k)*

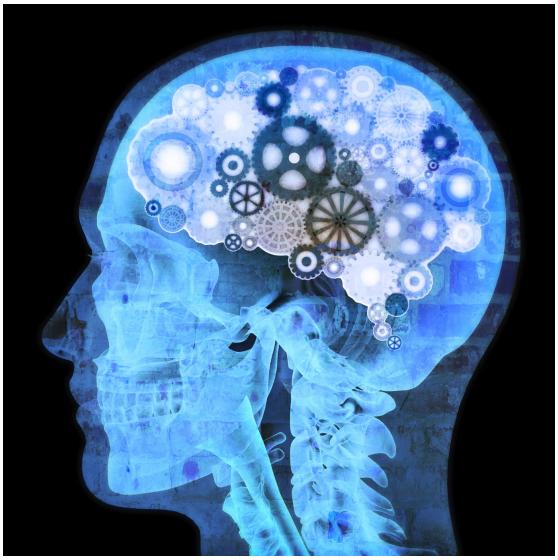
Iterate through each attribute of each object: *O(c)*

Obviously, these steps are run multiple-times throughout solve, but as these numbers approach infinity, addition in big-O goes to 0. So, if I had to estimate the worst-case run-time of my solve algorithm at one point, I would say it's *O(nkc)*, where *n* is the number of answer choices, *k* is the number of objects in each figure, and *c* is the number of attributes for each object. The run-time could become much worse if each figure will star to contain many more objects, and where each object has many attributes. Right now there are only 6 possible answer choices, but obviously this could be increased a swell, hindering the run-time.

This run-time could be improved by storing the data in hash tables and O(1) access variables, eliminating the amount of times that iteration is needed through each level.

6. Relation to Human Cognition

A. How “intelligent” is my agent?



Even though my agent is much more intelligent than my implementation from Project1, I would definitely say it has a very long way to go. In some ways it definitely mirrors human-like intelligence. It tries to make quick inferences about the problem and eliminate choices that obviously don’t pass preliminary cases. My agent very much uses problem reduction to analyze the goal situation and quickly eliminate the choices that don’t pass the initial screening. I feel like humans solve problems very much in this same way. For instance, any type of multiple choice exam, I try to eliminate as many answers as possible before even choosing a solution. This is natural as it’s hard to fully concentrate and break down many possible answers, but when working with a much smaller answer space, this becomes much easier. This works the same way for my agent: the more choices it can eliminate the higher the probability it has of choosing the right answer.

However, there are some situations where it just cannot compete with human intelligence. Pattern detection is something I greatly underestimated. The human eye is an amazing machine, able to instantaneously see the detailed difference between slight angle changes or movements, and know where the transformation came from. It’s extremely hard to teach an agent that it was a reflection of the y-axis and then a small turn to the right when just given numbers as input, but it’s no problem to see that. While having to design an artificial agent I have learned how much humans rely on pattern detection to solve problems.