CS7637: Project 1 (Fall 2015)

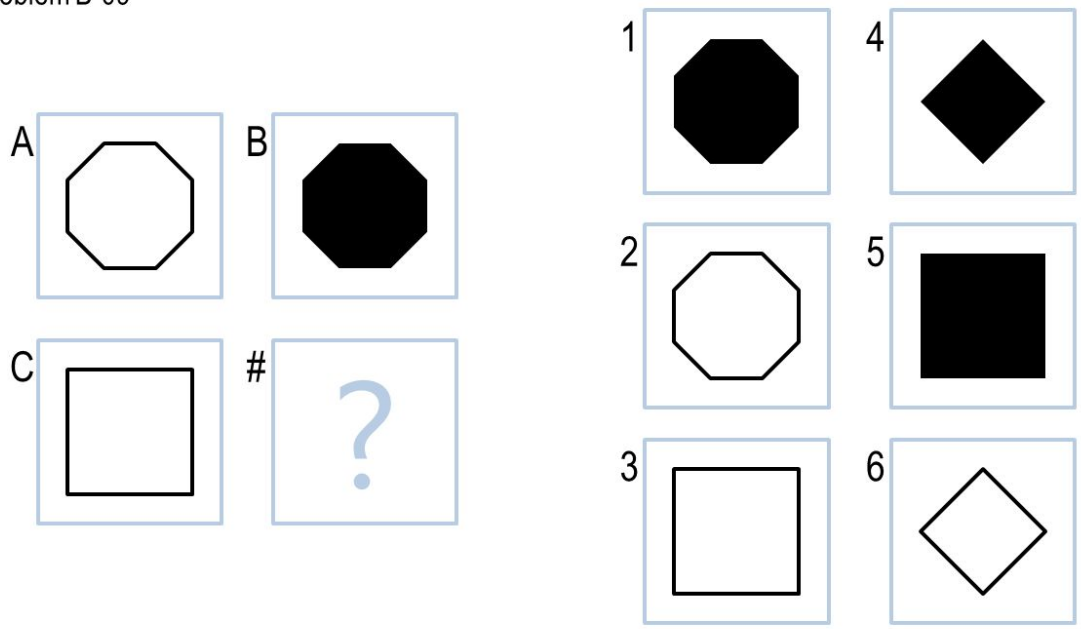
Due: September 2t7th at 11:59PM UTC-12 (Anywhere on Earth time)

- Getting Started
  - Goal
  - Overview
  - Getting Started
  - The Problems
- Working with the Code
  - The Code
  - The API
  - checkAnswer()
  - Libraries
  - Image Processing
- Your Submission
  - Submission
  - Execution
  - Efficiency
  - Project Reflection
  - Grading
  - Recovery Credit
  - Best Projects
- Helpful Tips
  - Sample Code
  - Common Submission Mistakes
  - Tips for Getting Started

Assignment: Solving 2x2 RPM using Verbal and/or Visual Representations

Make sure to read the more general project description here for a high-level introduction and description of the class project.

Basic Problem B-09



Getting Started

Goal

The goal of Knowledge-Based AI is to develop human-level, human-like intelligence. To that end, one way of evaluating the success of KBAI agents is to have them take human intelligence tests. In this project, you will develop agents that can address a specific intelligence test. In particular, in this project, your agents will solve 2x2 visual analogy problems using verbal and visual representations.

Overview

Design an agent that can answer 2x2 visual analogy problems based on both verbal and visual representations. You will have access to 24 sample problems to use in designing your agent: 12 Basic problems and 12 Challenge problems. After you submit your code, your agent will be run against these 24 problems, as well as 24 additional problems that you have not seen before: 12 Test problems and 12 Raven's problems. Your grade will be based on your agent's performance on the 12 Basic and 12 Test problems. The 12 Challenge and 12 Raven's problems will not impact your grade. You will also write a design report of roughly 1500 words describing the way your algorithm works, its relative strengths, and its relative weaknesses.

Getting Started

To get started, download the code package. The code package is available in two languages: Java and Python (compatible with either Python 2 or Python 3). You may choose to use either. Contained in the package are three things: the code, the API, and the sample problems. Note that the same code package can be used for all three projects in this course: the only difference is which problems your agent addresses on each project.

The Problems

The only difference between Projects 1, 2, and 3 is which problems your agent is run against. In Project 1, your agent is run against Problem Set B -- Basic, Test, Challenge, and Raven's. Your agent is only *graded* on its performance on Basic and Test -- Challenge and Raven's are used for your curiosity.

You are provided with the Basic and Challenge problems to use in designing your agent. The Test and Raven's problems are hidden and will only be used when grading your project. This is to test your agents for generality: it isn't hard to design an agent that can answer questions it has already seen, just as it would not be hard to score well on a test you have already taken before. However, performing well on problems you and your agent *haven't* seen before is a more reliable test of intelligence.

Your grade is based solely on your agent's performance on the Basic and Test problems. The Challenge and Raven's problems are optional. The Test problems are written to be directly and closely analogous to the Basic problems. For example, Test Problem B-03 will use similar transformations and reasoning to Basic Problem B-03. Thus, in designing your agent, you should rest assured that it will not be tested on anything radically different from what you have seen in the Basic problems. Similarly, you can use these direct comparisons to evaluate your agent's generality: if your agent correctly answers Basic Problem B-03 but misses on Test Problem B-03, that suggests its reasoning on these problems may be relatively brittle. Generally, your agent will first run on all available Basic problems, then all available Test problems, then all available Challenge problems, then all available Raven's problems.

Both the Basic and the Test problems are also closely analogous to the corresponding Raven's problem. Basic and Test Problems B-03 borrow the same reasoning from Raven's Problem B-03. The Raven's problems, however, often involve an additional layer of visual complexity, such as unrecognizable shapes, textures, or patterns. The Challenge problems are sometimes (but not always) written specifically to bridge the gap between Basic and Raven's problems. Although you are not graded on the Challenge and Raven's sets, if your agent actually manages to perform *better* on these sets than the Basic and Test sets, we may

award some extra points. After all, the ultimate goal is to perform well on the true Raven's test.

All problems are contained within the Problems folder of the downloadable. Problems are divided into sets, and then into individual problems. Each problem's folder has three things:

- The problem itself, for your benefit.
- A ProblemData.txt file, containing information about the problem, including its correct answer, its type, and its verbal representation (if applicable).
- Visual representations of each figure, named A.png, B. png, etc.

You should not attempt to access ProblemData.txt directly; its filename will be changed when we grade projects. Generally, you need not worry about this directory structure; all problem data will be loaded into the RavensProblem object passed to your agent's Solve method, and the filenames for the different visual representations will be included in their corresponding RavensFigures.

Working with the Code

The Java code is available here; the Python code is available here. Note that these code packages contain a file instructing your agent to only look at those problems relevant for Project 1; updated versions of this file will be supplied for Project 2 and Project 3.

The Code

The downloadable package has a number of either Java or Python files: RavensProject, ProblemSet, RavensProblem, RavensFigure, RavensObject, and Agent. Of these, you should only modify the Agent class. You may make changes to the other classes to test your agent, write debug statements, etc. However, when we test your code, we will use the original versions of these files as downloaded here. Do not rely on changes to any class except for Agent to run your code. In addition to Agent, you may also write your own additional files and classes for inclusion in your project.

In Agent, you will find two methods: a constructor and a Solve method. The constructor will be called at the beginning of the program, so you may use this method to initialize any information necessary before your agent begins solving problems. After that, Solve will be called on each problem. You should write the Solve method to return its answer to the given question:

- 2x2 questions have six answer options, so to answer the question, your agent should return an integer from 1 to 6.
- 3x3 questions have eight answer options, so your agent should return an integer from 1 to 8.

- If your agent wants to skip a question, it should return a negative number. Any negative number will be treated as your agent skipping the problem.

You may do all the processing within Solve, or you may write other methods and classes to help your agent solve the problems.

When running, the program will load questions from the Problems folder. It will then ask your agent to solve each problem one by one and write the results to ProblemResults.csv. You may check ProblemResults.csv to see how well your agent performed. You may also check SetResults.csv to view a summary of your agent's performance at the set level.

The API

Included in the downloadable is the API for interacting with the code (API/index.html in the downloadable). You may use this and the in-line comments to understand the structure of the problems. Briefly, however:

- RavensProject: The main driver of the project. This file will load the list of problem sets, initialize your agent, then pass the problems to your agent one by one.
- Agent: The class in which you will define your agent. When you run the project, your Agent will be constructed, and then its Solve method will be called on each RavensProblem. At the end of Solve, your agent should return an integer as the answer for that problem (or a negative number to skip that problem).
- ProblemSet: A list of RavensProblems within a particular set.
- RavensProblem: A single problem, such as the one shown earlier in this document. This is the most complicated and important class in the project, so let's break it into parts. RavensProblem includes:
    - A HashMap (Java) or Dictionary (Python) of the individual Figures (that is, the squares labeled "A", "B", "C", "1", "2", etc.) from the problem. The RavensFigures associated with keys "A", "B", and "C" are the problem itself, and those associated with the keys "1", "2", "3", "4", "5", and "6" are the potential answer choices.
    - A String representing the name of the problem and a String representing the type of problem ("2x2" or "3x3").
    - Variables hasVisual and hasVerbal indicating whether that problem has a visual or verbal representation (all problems this semester have visual representations, only some have verbal representations).
    - The correct answer and your agent's answer to the problem. The correct answer can only be accessed via the checkAnswer method, which requires your agent to pass an answer as a parameter. After calling checkAnswer, your agent will not be able to change its answer. Thus, your agent can only see the correct

answer after already giving its answer. Your agent can therefore learn from its mistakes and apply that learning to future problems.

- RavensFigure: A single square from the problem, labeled either "A", "B", "C", "1", "2", etc. All RavensFigures have a filename referring to the visual representation (in PNG form) of the figure's contents. Problems with verbal representations also contain dictionaries of RavensObjects. In the example above, the squares labeled "A", "B", "C", "1", "2", "3", "4", "5", and "6" would each be separate instances of RavensFigure, each with a list of RavensObject.
- RavensObject: A single object, typically a shape such as a circle or square, within a RavensFigure. For example, in the problem above, the Figure "C" would have one RavensObject, representing the square in the figure. RavensObjects contain a name and a dictionary of attributes. Attributes are key-value pairs, where the key is the name of some general attribute (such as 'size', 'shape', and 'fill') and the value is the particular characteristic for that object (such as 'large', 'circle', and 'yes). For example, the square in figure "C" could have three RavensAttributes: shape:square, fill:no, and size:very large. Generally, but not always, the representation will provide the shape, size, and fill attributes for all objects, as well as any other relevant information for the particular problem.

The API is ultimately somewhat straightforward, but it can be complicated when you're initially getting used to it. The most important things to remember are:

- Every time Solve is called, your agent is given a single problem. By the end of Solve, it should return an answer as an integer. You don't need to worry about how the problems are loaded from the files, how the problem sets are organized, or how the results are printed. You need only worry about writing the Solve method, which solves one question as a time.
- RavensProblems have a dictionary of RavensFigures, with each Figure representing one of the squares of the problems. All RavensFigures have filenames so your agent can load the PNG with the visual representation. If the problem has a verbal representation as well (hasVerbal or hasVerbal() is true), then each RavensFigure has a dictionary of RavensObjects, each representing one shape in the Figure (such as a single circle, square, or triangle). Each RavensObject has a dictionary of attributes, such as "size":"large", "shape":"triangle", and "fill":"yes".

checkAnswer()

A key aspect of Knowledge-Based AI is learning, but learning is facilitated by feedback. It would be difficult for your agent to learn if it did not know what problems it had gotten right or wrong in the past. While it's completing the problems, your agent has the option to check its answers using the checkAnswer method in RavensProblem. The checkAnswer method

requires that the agent give an answer, and in return, the checkAnswer method returns the correct answer. Once your agent has called checkAnswer on a problem, it cannot change its answer to the problem; in other words, your agent cannot change its answer after seeing the correct answer. However, it can use the knowledge of the correct answer to change its own reasoning so that it has a better chance of getting future problems correct.

Note that using the checkAnswer method is optional. The Solve method should always return your agent's answer to the problem. If your agent called checkAnswer before returning an answer at the end of Solve, however, the answer passed to checkAnswer will be used in scoring.

Note to Python Users
Because Python does not support private variables, it is programmatically possible to directly access and modify the correctAnswer and givenAnswer variables in RavensProblem.py. Do not access these variables directly. When we grade your code, the names of these variables will be changed, and your code will not run if you attempt to access these variables directly. Instead, only access these variables through the checkAnswer method and by returning your agent's answer to each problem at the end of the Solve method.

Libraries

No external libraries are permitted in Java. In Python, the **only** permitted library is the latest version of the Python image processing library Pillow. For installation instructions on Pillow, see this page. No other libraries are permitted.

Image Processing

Generally, we do not allow external libraries. For Java, you may use anything contained within the default Java 8 installation. Java 8 has plenty of image processing options. We recommend using BufferedImage, and we have included a bit of sample code below for loading images into BufferedImage. If you have other suggestions, please bring them up on Piazza!

Python has no native support for image processing, so an external library must be used. The **only** external library we support for Python is Pillow. You can install pillow simply by running `easy_install pillow`. More comprehensive information on installing Pillow can be found here. We have included a code segment below on loading an image from a file with Pillow.

Your Submission

Submission

In order to ensure smooth grading, it is crucial that you submit your project according to the following instructions. Points may be taken off if these instructions are not followed.

1. Zip up your Agent.java or Agent.py file, along with all other files you have created (not the files supplied to you at the beginning). The .py and .java files should be in the root of the .zip file, although other .py and .java files may be in subfolders if necessary.
2. Name the zip according to the following convention: Project1-{gtusername}-{language}.zip. For example, my submission (if I used Python 2) would be Project1-djoyner3-Python2.zip. Python users: **Make sure** to include your Python version, either Python2 or Python3.
3. Upload the .zip file as your submission for Project 1.
4. Separately, upload your project reflection as a PDF as your submission for Project 1 - Project Reflection.

Execution

Projects are graded using a script while project reflections are hand-graded by our teaching team. In order to be executable by this script, your agent must follow the format above. To test out whether or not your agent is script-ready, here are the details for how we will run your code. Before submitting, please try this out on your own using a clean download of the original project files (except for Agent). If your code does not run with these commands, we may have to take points off.

Python Execution

If you use Python 2 or 3, we will unzip your submission, then copy the original files (besides Agent.py) into your folder, overwriting any files you may have submitted with those filenames. We will then run your code using PyLauncher, using the line: `py -2 RavensProject.py` or `py -3 RavensProject.py`.

Java Execution

If you use Java, we will unzip your submission into a new ravensproject folder, then copy the original files (besides Agent.java) into your folder, overwriting any files you may have submitted with those filenames. We will then copy in the Problems folder alongside the ravensproject folder. We will compile your code using dir /s /B *.java > sources.txt followed by `javac @sources.txt`, then run your agent in the project by calling `java ravensproject.RavensProject`. You should, thus, *not* submit the ravensproject folder

itself, but rather those files inside it. If your agent relies on packages to execute properly, make sure to zip up your code in the proper folder structure.

Efficiency

Efficiency can be a major concern with these agents; some problems can require reasoning that takes an enormous amount of time. It is acceptable for your agent to take a few minutes to address the problems, but it should not take significantly longer than that. We may cut your agent off if it goes over 15 minutes to address the test as a whole. However, your agent should show some signs of progress to show that it is not stuck in an infinite loop. For example, you may use print statements to output when each problem has been solved or when a step has been successfully completed.

Project Reflection

In addition to completing your agent, you are also asked to complete a project reflection of roughly 1500 words. The project reflection serves two purposes: (a) to help you reflect on and learn from your experience during the project, and (b) to help communicate your ideas to your peers in the class as well as the graders.

In your project reflection, you should answer the following questions. You can separate your project reflection into multiple sections each answering a question, or you can write a more general project reflection that covers these questions:

- 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?
- 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?
- 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?
- 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?
- 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?
- 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?
- 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?

As each project builds on the previous one, it is likely that much of your information will be the same from project to project. In this event, what is important is to remember the learning goals: this is a reflection, and you are meant to reflect on the progress since the previous project. When applicable, it's fine to give only a short description of your previous project and spend most of your time focusing on your progress on the new project. Please mention when you're doing so, though, so that the graders and your peers know when you're intentionally summarizing rather than skipping information.

Your project reflection will be evaluated on a scale of 0 to 35. Each of the above questions will be evaluated on a scale of 0 to 5 to determine your score. As with other assignments, a 90% should not be considered the threshold for an 'A' on the project reflection -- make sure to check the stats posts when grades are posted to have context for your grade.

Grading

Your grade will be based on three components:

- Your agent's score on the 12 problems in Basic Problems B (20% of your grade).
- Your agent's score on the 12 problems in Test Problems B (40% of your grade).
- Your project reflection (40% of your grade).

In calculating your agent's score on the Basic and Test problems, your agent will receive 1 point for each correct response. Your agent will also lose 1/5th of a point for each incorrect response, and lose nothing for each skipped problem. To skip a problem, your agents should give a negative number as its answer. This is to correct for the effect of randomness, as well as to encourage you to equip your agent with some metacognition to decide how confident it is on a given answer and whether it is worth answering.

In addition to these grading criteria, your agent may be awarded some extra points if its performance on the Challenge and Raven's problems outperforms its performance on the Basic and Test problems. Your agent may also be awarded some extra points if we regard your agent's approach as particularly novel and unique. Check out the Take Chances, Make Mistakes section of the Overall Project Guidelines for more on this.

Note that many of the problems are very difficult and your agent is certainly not expected to solve all of them. The current state of the art in this field still only answers 70% or so of the problems correct. Thus, a 90/100 should not be considered the threshold for an A. Grades generally will be normalized after the fact based on the class's performance as a whole, so make sure to read the announcements accompanying the grades to understand how to interpret your grade, and don't freak out when you see a score that would usually translate to a C or worse (last semester, the class average on the last project was a 65). We will also be more generous in our interpretation of Project 1 since there's a larger learning curve and the standards for success, both in this class and for these problems, has not yet been set.

Recovery Credit

The ultimate goal of this project is to design an agent that can perform well on all 192 problems. Thus, your submissions for each project will run on the previous projects' problems as well; Project 2 will run on sets B and C, and Project 3 will run on all four sets, B, C, D, and E.

Previously, we graded each agent's performance on all these problems. However, students in the past have pointed out that penalizes students who did poorly on Project 1 -- their agent is running against the same problems, and so their grade is already lower than others who did better on Project 1. So, this semester, we're revising this so that it can only help you. If your agent performs better on problem set B in Project 2 than in Project 1, you will receive half credit back. So, if you get a score of 4 on Basic Problem Set B in Project 1, and a score of 8 on Basic Problem Set B in Project 2, then your Project 1 grade will be based on a score of 6 on this set.

For more on this, please see the Repeated Problem Set section of the Overall Project Guidelines.

Best Projects

At the conclusion of each project, a handful of the best projects will be selected and, with the students' permission, posted for public viewing. The selection of the "best" will be made in large part based on how many problems each student's agent gets correct, but it may also be based partially on subjective analysis by the graders. If a particular project takes a particularly unique approach, for example, it may be selected as an exemplary project even if other projects technically performed better.

Helpful Tips

Sample Code

To help you get started, here's a couple little code bits:

Java

To iterate over all the RavensFigures in a RavensProblem:

```
for(String figureName : problem.getFigures().keySet()) {
	RavensFigure thisFigure = problem.getFigures().get(figureName);
	...
}
```

To iterate over all the Objects in a RavensFigure (given figureName as above):

```
RavensFigure thisFigure = problem.getFigures().get(figureName);

for(String objectName : thisFigure.getObjects().keySet()) {
	RavensObject thisObject = thisFigure.getObjects().get(objectName);
	...
}
```

To iterate over all the attributes in a RavensObject (given objectName as above):

```
RavensObject thisObject = thisFigure.getObjects().get(objectName);

for(String attributeName : thisObject.getAttributes().keySet()) {
	String attributeValue = thisObject.getAttributes().get(attributeName);
	...
}
```

To load a visual representation from a file into a BufferedImage:

```
import java.awt.Image;
import java.io.File;
import javax.imageio.ImageIO;

…

RavensFigure figureA = problem.getFigures().get("A");
try { // Required by Java for ImageIO.read
	Image figureAImage = ImageIO.read(new File(figureA.getVisual()));
	...
} catch(Exception ex) {}
```

After loading a visual representation from a file into a BufferedImage, to iterate over all pixels in the image:

```
for(int i = 0 ; i < figureAImage.getWidth() ; i++) {
	for(int j = 0 ; j < figureAImage.getHeight() ; j++) {
		int thisPixel = figureAImage.getRGB(i,j);
		...
	}
}
```

Python

To iterate over all the RavensFigures in a RavensProblem:

```
for figureName in problem.figures:
      thisFigure = problem.figures[figureName]
      ...
```

To iterate over all the Objects in a RavensFigure (given figureName as above):

```
thisFigure = problem.figures[figureName]

for objectName in thisFigure.objects:
      thisObject = thisFigure.objects[objectName]
```

To iterate over all the attributes in a RavensObject (given objectName as above):

```
thisObject = thisFigure.objects[objectName]

for attributeName in thisObject.attributes:
      attributeValue = thisObject.attributes[attributeName]
```

To load a visual representation from a file into a Image (from Pillow):

```
from PIL import Image

...

figureA = problem.figures['A']
figureAImage = Image.open(figureA.visualFilename)
```

After loading a visual representation from a file into a BufferedImage, to iterate over all pixels in the image:

```
figureALoaded = figureAImage.load()
for i in range(0, figureAImage.size[0]):
      for j in range(0, figureAImage.size[1]):
            thisPixel = figureALoaded[i, j]
```

Common Submission Mistakes

Over the past two semesters of running this class, certain mistakes have come up several times. Make sure not to make these to avoid losing points on your project:

- Not properly naming your .zip file submission. Remember, it should be of the form: Project1-{gtusername}-{language}.zip. If I used Python3, that would be Project1-djoyner3-Python3.zip. If you use Python, **make sure** to specify which version of Python you're using.
- Java is generally backwards compatible. However, we have had a couple isolated instances where a student's projects performed better under Java 7 than Java 8, or

vice versa. We run all projects under Java 8, so make sure you're running your project under Java 8 to have the right expectation of its success.

- In Java, make sure any new files you create have the right package name associated with them. All new files in the core folder, for instance, should be in the ravensproject package.
- In Java, zip up the *contents* of the ravensproject folder, not the ravensproject folder itself.
- Double-check and make sure you haven't commented out any important code, or left any experimental code in your submission.

Tips for Getting Started

This project can be a bit overwhelming at first. Don't get discouraged! We've heard time and time again students report that they had no clue how they were going to succeed at first, but by the end of it they had a solid grasp of the concepts and workflows necessary. Note in the Overall Project Guidelines the note on authenticity. This project can seem overwhelming to you because it's a big, open question facing the AI community today. You're working on a real problem.

After two semesters and almost 500 students, a few common tips have started to emerge for how to get started on the project. These tips aren't meant to pigeonhole you for the entire duration of the project, but they're meant to help you overcome that initial hump and get something working. You're free to ignore these; these are mostly supplied in case you're having trouble getting started.

- Instead of trying to design an agent that can answer every problem right from the beginning, try instead to write an agent that can solve one problem. Then, look at why that approach is failing on a second problem, and see if you can tweak it to get that second problem right. Continue that iteration and you'll come to an idea for a broader plan, but you'll also have an agent that's already partially successful.
- Look for common problem types or feasible heuristic approaches. Are there three problems that use similar transformations? Try to focus on those to get the most progress for the time you invest.
- Remember, you're only graded on Basic and Test problem sets. Test problem sets are written to be very, very closely analogous to the Basic problem set, so don't let yourself get distracted by the Challenge problems. Those are provided for those that want to really get into the project and prepare for the real Raven's problems, but they shouldn't concern you if you're having trouble.