Project Overview (Fall 2022)

Our semester-long class project in involves constructing an AI agent to address a human intelligence test. The project is due at the end of the semester, but there are a number of required milestones to pass along the way. These are (a) to ensure that you are getting an early enough start to have a chance for success and (b) to give you opportunities to see your classmates' approaches and possibly incorporate their ideas into your own project.

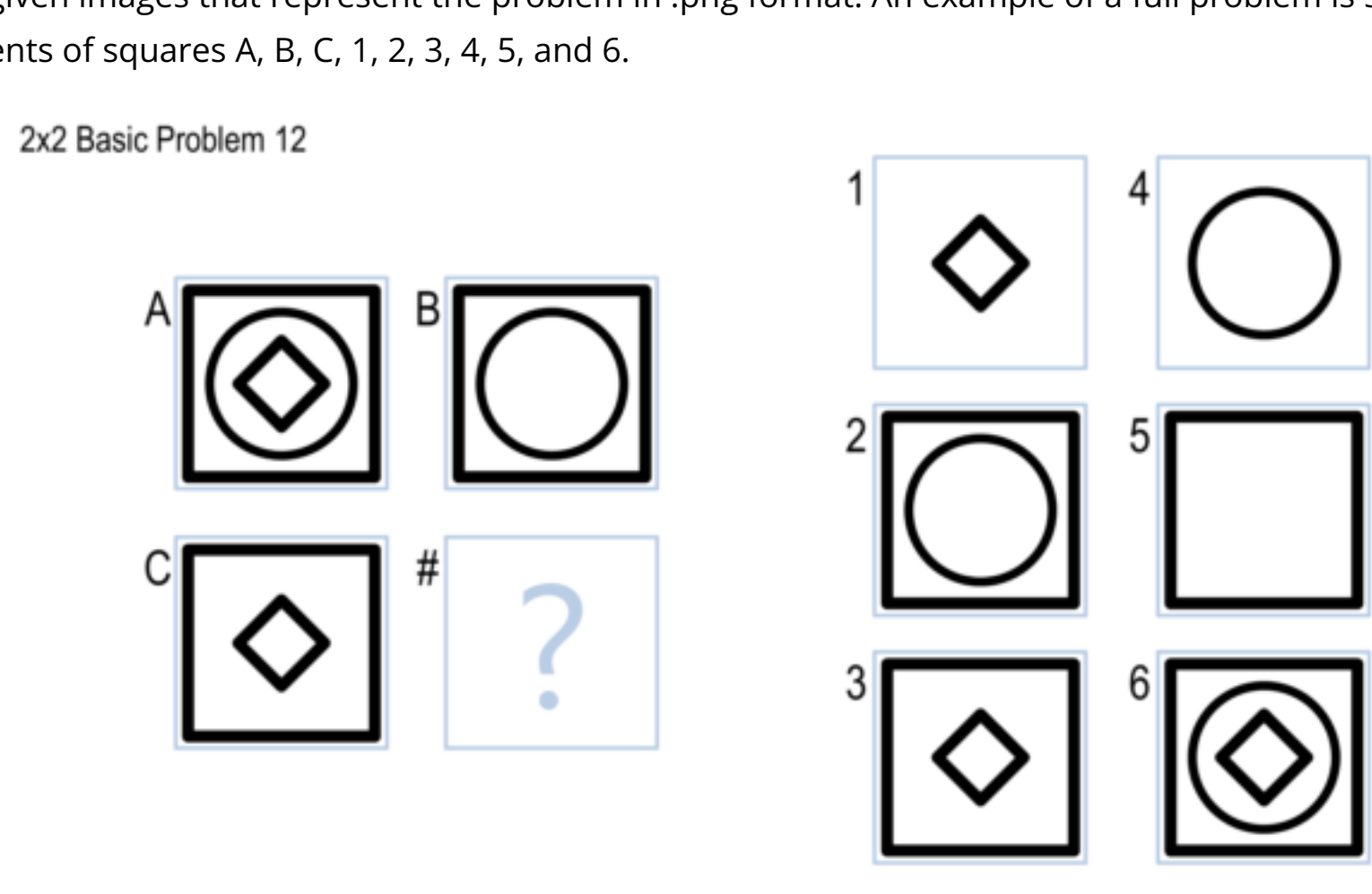
This page covers the project as a whole, emphasizing what your end-goal is for the end of the semester.

In a (Large) Nutshell

The CS7637 class project is to create an AI agent that can pass a human intelligence test. You'll download a code package that contains the boilerplate necessary to run an agent you design against a set of problems inspired by the Raven's Progressive Matrices test of intelligence. Within it, you'll implement the Agent.py file to take in a problem and return an answer.

There are four sets of problems for your agent to answer: B, C, D, and E. Each set contains four types of problems: Basic, Test, Challenge, and Raven's. You'll be able to see the Basic and Challenge problems while designing your agent, and your grade will be based on your agent's answers to the Basic and Test problems. The milestones throughout the semester will carry you through tackling more and more advanced problems: for Milestone 1, you'll just familiarize yourself with the submission process and data structures. For Milestone 2, you'll target the first set of problems, the relatively easy 2x2 problems from Set B. For Milestone 3, you'll move on to the second set of problems, the more challenging 3x3 problems from Set C. For Milestone 4, you'll look at the more difficult Set D and Set E problems, building toward the final deliverable a bit later.

For all problems, your agent will be given images that represent the problem in .png format. An example of a full problem is shown below; your agent would be given separate files representing the contents of squares A, B, C, 1, 2, 3, 4, 5, and 6.



Don't worry if the above doesn't make sense quite yet — the projects are a bit complex when you're getting started. The goal of this section is just to provide you with a high-level view so that the rest of this document makes a bit more sense.

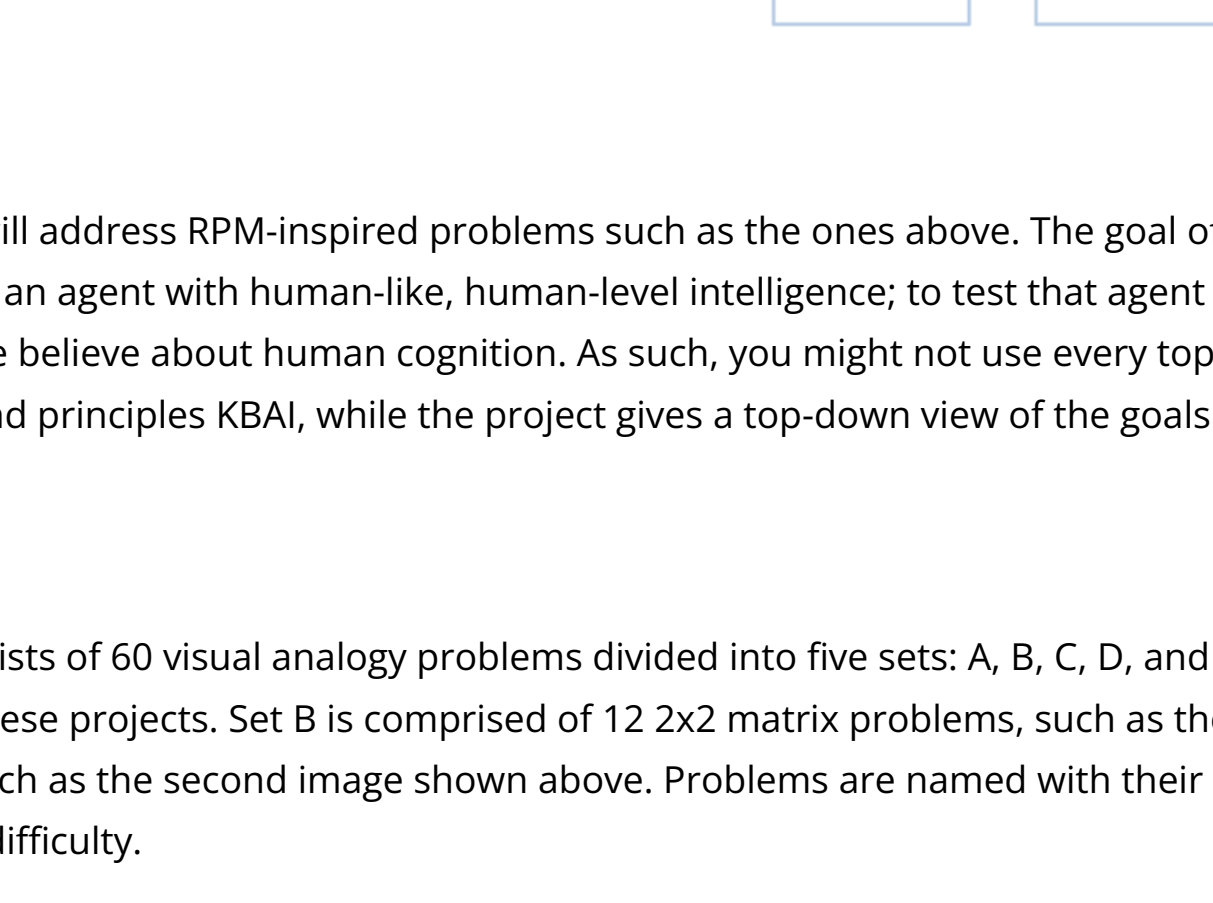
Background and Goals

This section covers the learning goals and background information necessary to understand the projects.

Learning Goals

One goal of Knowledge-Based Artificial Intelligence is to create human-like, human-level intelligence, and to use that to reflect on how humans actually think. If this is the goal of the field, then what better way to evaluate intelligence of an agent than by having it take the same intelligence tests that humans take?

There are numerous tests of human intelligence, but one of the most reliable and commonly-used is Raven's Progressive Matrices. Raven's Progressive Matrices, or RPM, are visual analogy problems where the test-taker is given a matrix of figures and asked to select the figure that completes the matrix. An example of a 2x2 problem was shown above; an example of a 3x3 problem is shown below.



In these projects, you will design agents that will address RPM-inspired problems such as the ones above. The goal of this project is to authentically experience the overall goals of knowledge-based AI: to design an agent with human-like, human-level intelligence; to test that agent against a set of authentic problems; and to use that agent's performance to reflect on what we believe about human cognition. As such, you might not use every topic covered in KBAI on the projects; the topics covered give a bottom-up view of the topics and principles KBAI, while the project gives a top-down view of the goals and concepts of KBAI.

About the Test

The full Raven's Progressive Matrices test consists of 60 visual analogy problems divided into five sets: A, B, C, D, and E. Set A is comprised of 12 simple pattern-matching problems which we won't cover in these projects. Set B is comprised of 12 2x2 matrix problems, such as the first image shown above. Sets C, D, and E are each comprised of 12 3x3 matrix problems, such as the second image shown above. Problems are named with their set followed by their number, such as problem B-05 or C-11. The sets are of roughly ascending difficulty.

For copyright reasons, we cannot provide the real Raven's Progressive Matrices test to everyone. Instead, we'll be giving you sets of problems — which we call "Basic" problems — inspired by the real RPM to use to develop your agent. Your agent will be evaluated based on how well it performs on these "Basic" problems, as well as a parallel set of "Test" problems that you will not see while designing your agent. These Test problems are directly analogous to the Basic problems; running against the two sets provides a check for generality and overfitting. Your agents will also run against the real RPM as well as a set of Challenge problems, but neither of these will be factored into your grade.

Overall, by the end of the semester, your agent will answer 192 problems. More on the specific problems that your agent will complete are in the sections that follow.

Each problem set (that is, Set B, Set C, Set D, and Set E) consists of 48 problems: 12 Basic, 12 Test, 12 Raven's, and 12 Challenge. Only Basic and Test problems will be used in determining your grade. The Raven's problems are run for authenticity and analysis, but are not used in calculating your grade.

In designing your agent, you will have access to the Basic and Challenge problems; you may run your agent locally to check its performance on these problems. You will not have access to the Test or Raven's problems while designing and testing your agent: when you upload your agent to Gradescope, you will see how well it performs on those problems, but you will not see the details of the problems themselves. Challenge and Raven's problems are not part of your grade.

Note that the Challenge problems will often be used to expose your agent to extra properties and shapes seen on the real Raven's problems that are not covered in the Basic and Test problems. The problems themselves generally ascend in difficulty from set to set (although many people reflect that Set E is a bit easier than Set D.

Finally, note that when submitting your code, Gradescope randomizes the order of possible answers. For example, for the first problem listed earlier on this page, the answer is 5, the plain square. When you submit to Gradesope, the plain square will be randomly assigned a different number: it could be 1, 2, 3, 4, 5, or 6. The content of the image is the same, only the number differs. This is to prevent agents that overfit to the answers: you cannot, for example, write an agent that knows, "The answer to problem B-11 is 5" because when you submit, the number is randomly changed.

Details & Deliverables

This section covers the more specific details of the four project milestones, as well as the final project you will submit.

Project Milestones

Your ultimate goal is to submit a final project that attempts all 192 problems. However, to help ensure that you start early and to give you an opportunity to see and learn from your classmates' approaches, there are four intermediate milestones. On each of these milestones, your agent will only run against a subset of the full set of problems to allow you to test more efficiently. You will also write a brief report on your current approach for each milestone; the primary purpose of these reports will be to help you get feedback from classmates and see their approaches. For each milestone, you will be graded on a combination of your agent's performance and the report that you write; the bars for performance on the milestones are relatively low, however, as the goal is to ensure that you are getting started early.

Each milestone has its own page. In brief, however:

- Milestone 1: Set B, Basic Problems only. The goal of this milestone is simply to ensure you've set up your local project infrastructure and familiarized yourself with Gradescope. You will receive 100% of your performance credit as long as your agent answers *any* problem correctly. Your report will focus on early ideas you have for approaching the project.
- Milestone 2: Set B, all problems. The goal of this milestone is to ensure you have started on the early, easier problems early in the semester. As long as your agent can answer 5 (out of 12) Basic B and 5 (out of 12) Test B problems correctly, you will receive full performance credit.
- Milestone 3: Set C. The goal of this milestone is to ensure you have generalized your approach out to the more difficult 3x3 problems by an appropriate time of the semester. As long as your agent can answer 5 (out of 12) Basic C and 5 (out of 12) Test C problems correctly, you will receive full performance credit.
- Milestone 4: Sets D and E. The goal of this milestone is to ensure you have looked at all four sets before the final project deadline, so that you may spend the last portion of the semester refining, improving, and writing your final report. As long as your agent can answer 10 (out of 24) Basic D & E and 10 (out of 24) Test D & E problems, you will receive full performance credit.

For each milestone, your code must be submitted to the autograder by the deadline. However, it is okay if your project is still running after the deadline. Note that Gradescope by default counts your *last* submission for a grade; if you want to count an earlier submission, you *must* activate that earlier submission.

On each milestone, your grade will be 50% meeting the performance expectations and 50% the report you write up. You will submit your agent to Gradescope and your report to Canvas as a PDF. The four milestones together are 15% of your course grade; each is thus 3.75% of your course grade.

Final Project

The final project will run against all 192 problems. You can submit to the final project throughout the semester to see how your agent is doing so far, but you should make sure to submit to the Milestone submissions as well.

More information about the final project is available on the final project page. For the final project, you will write a longer, more formal, and more complete report on your project. Your score will be based on raw performance on the Basic and Test problems. Like the milestones, performance will be 50% of your grade and your report will be 50% of your grade. Your final project is 15% of your course grade.

Getting Started

To make it easier to start the project and focus on the concepts involved (rather than the nuts and bolts of reading in problems and writing out answers), you'll be working from an agent framework in Python. You can get the framework in one of two ways:

- Clone it from the master repository with 'git clone –recurse-submodules https://github.gatech.edu/omscs7637/RPM-Project-Code.git'
- Download RPM-Project-Code as a zip file. This method allows you to obtain the code if you are having trouble accessing the Georgia Tech Github site.

You will place your code into the Solve method of the Agent class supplied. You can also create any additional methods, classes, and files needed to organize your code; Solve is simply the entry point into your agent.

The Problem Sets

As mentioned previously, by the final project, your agent will run against 192 problems: 4 Sets of 48 problems. Each of the 4 Sets is broken down into four subsets of 12: 12 Basic, 12 Test, 12 Raven's, and 12 Challenge.

You can see the Basic and Challenge problems and test your agent's performance on them locally. You cannot see the Test and Raven's problems, and your agent's performance will only be tested when you submit to Gradescope. Your grade will be based solely on the Basic and Test problems.

The Raven's problems are used so that you can see how your agent is performing on the real Raven's test. The Challenge problems are primarily there to expose your agent to certain details that are present in the Raven's problems but not in the Basic problems (such as shapes shaded with diagonal lines).

Within each set, the Basic, Test, and Raven's problems are constructed to be roughly analogous to one another. The Basic problem is constructed to mimic the relationships and transformations in the corresponding Raven's problem, and the Test problem is constructed to mimic the Basic problem very, very closely. So, if you see that your agent gets Basic problem B-05 correct but Test and Raven's problems B-05 wrong, you know that might be a place where your agent is either overfitting or getting lucky. This also means you can anticipate your agent's performance on the Test problems relatively well: each Test problem uses a near-identical principle to the corresponding Basic problem. In the past, agents have averaged getting 85% as many Test problems right as Basic problems, so there's a pretty good correlation there *if* you're using a robust, general method.

The Problems

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.

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 and its type.
- 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 framework code is available under Getting Started above. You may modify ProblemSetList.txt to alter what problem sets your code runs against locally; this will be useful early in the term when you probably do not need to bother thinking about later problem sets yet. This will not affect what it runs against on Gradescope.

The Code

The downloadable package has a number of Python files: RavensProject, ProblemSet, RavensProblem, RavensFigure, 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 Documentation

- 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.
- RavensGrader: The grading file for the project. After your agent generates its answers, this file will check the answers and assign a score.
- 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. A RavensProblem includes:
  - A Dictionary 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").
- RavensFigure: A single square from the problem, labeled either "A", "B", "C", "1", "2", etc., containing a filename referring to the visual representation (in PNG form) of the figure's contents

The documentation 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 at a time.
- RavensProblems have a dictionary of RavensFigures, with each Figure representing one of the image squares in the problem and each key representing its letter (squares in the problem matrix) or number (answer choices). All RavensFigures have filenames so your agent can load the PNG with the visual representation.

Libraries

The permitted libraries for this term's project are:

- The Python image processing library Pillow (version 8.1.0). For installation instructions on Pillow, see this page.
- The latest version of the Numpy library (1.19.1 at time of writing). For installation instructions on numpy, see this page.
- A recent version of OpenCV (4.2.0, opencv-contrib-python-headless). For installation instructions, see this page.

Additionally, we use Python 3.8.0 for our autograder.

Submitting Your Code

This class uses Gradescope, a server-side autograder, to evaluate your submission. This means you can see how your code is performing against the Test and Raven's problems even without seeing the problems themselves. You will have access to separate areas to submit against the Milestone checks and to submit for the final project.

Submitting

To get started submitting your code, go to Canvas and click Gradescope on the left sidebar. Then, click CS7637 in the page that loads.

You will see five project options: Milestone 1, Milestone 2, Milestone 3, Milestone 4, Final Project, and Final Project.

Milestone 1 will run your code only against the Basic B problems. Milestone 2 will run your code only against the Basic B, Test B, Challenge B, and Raven's B problems. Milestone 3 will run your code against the Basic C, Test C, Challenge C, and Raven's C problems. Milestone 4 will run your code against the Basic D+E, Test D+E, Challenge D+E, and Raven's D+E problems.

To submit your code, drag and drop your project files (Agent.py and any support files *you* created; you should not submit the other files that we supplied) into the submission window. You may also zip your code up and upload the zip file. You'll receive a confirmation message if your submission was successful; otherwise, take the corrective action specified in the error message and resubmit.

Remember as noted above that when submitting to Gradescope, the answers will be shuffled. For example, the answer to the first problem shown on this page will always be the plain square, but it would not always be answer #5. If you find your agent is getting questions wrong on Gradescope that it answers correctly locally, it is likely that your agent is in part sensitive to the order of answers (e.g. "Pick the first answer that…") or sensitive to ordering factors that may change in a different environment (e.g. the order of dictionary keys when iterating over a dictionary); it is not that the answers themselves are different on Gradescope.

You may submit up to 40 times. The large majority of students do not need nearly that many submissions, so do not feel like you should use all 40; this cap is in place primarily to prevent brute force methods for farming information about patterns in hidden test cases or submitting highly random agents hoping for a lucky submission. Note that Gradescope has no way for us to increase your individual number of submissions, so we cannot return submissions to you in the case of errors or other issues, but you should have more than enough submissions to handle errors if they arise.

Getting Your Results

Then, wait for the autograder to finish. If there are no errors, you will see a detailed summary of your results. Each row in the test result output is the result of a single problem from the problem set: you'll see a score of Pass label if your agent answered the question correctly, and a Fail label if your agent did not answer the question correctly. Your overall score can be found at the top right of the screen and in the summary at the bottom of the test results.

If you'd like to resubmit your code, click "Resubmit" found at the bottom-right corner of the autograder results page. You can analyze and compare your submission results using the "Submission History" button on this page, too.

Selecting Your Results

Once you have made your last submission, click Submission History, and then click Active next to your best submission. This is the only way to commit your final score to Canvas and get points; Gradescope does not select your best score automatically. You **must** do this to receive points for your submission.

After the deadline, we will import your score to Canvas to use in calculating your final score on that milestone or the project.

Relevant Resources

Goel, A. (2015). Geometry, Drawings, Visual Thinking, and Imagery: Towards a Visual Turing Test of Machine Intelligence. In *Proceedings of the 29th Association for the Advancement of Artificial Intelligence Conference Workshop on Beyond the Turing Test.* Austin, Texas.

McGreggor, K., & Goel, A. (2014). Confident Reasoning on Raven's Progressive Matrices Tests. In *Proceedings of the 28th Association for the Advancement of Artificial Intelligence Conference.* Québec City, Québec.

Kunda, M. (2013). Visual problem solving in autism, psychometrics, and AI: the case of the Raven's Progressive Matrices intelligence test. Doctoral dissertation.

Emruli, B., Gayler, R. W., & Sandin, F. (2013). Analogical mapping and inference with binary spatter codes and sparse distributed memory. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE.

Little, D., Lewandowsky, S., & Griffiths, T. (2012). A Bayesian model of rule induction in Raven's progressive matrices. In *Proceedings of the 34th Annual Conference of the Cognitive Science Society*. Sapporo, Japan.

Kunda, M., McGreggor, K., & Goel, A. K. (2012). Reasoning on the Raven's advanced progressive matrices test with iconic visual representations. In *34th Annual Conference of the Cognitive Science Society*. Sapporo, Japan.

Lovett, A., & Forbus, K. (2012). Modeling multiple strategies for solving geometric analogy problems. In *34th Annual Conference of the Cognitive Science Society*. Sapporo, Japan.

Schwering, A., Gust, H., Kühnberger, K. U., & Krumnack, U. (2009). Solving geometric proportional analogies with the analogy model HDTP. In *31st Annual Conference of the Cognitive Science Society*. Amsterdam, Netherlands.

Joyner, D., Bedwell, D., Graham, C., Lemmon, W., Martinez, O., & Goel, A. (2015). Using Human Computation to Acquire Novel Methods for Addressing Visual Analogy Problems on Intelligence Tests. In *Proceedings of the Sixth International Conference on Computational Creativity*. Provo, Utah.

…and many more!