In this project we will... Hunt the Wumpus! The objective is to build an agent that can explore the Wumpus World, combining its perceptions of the world with its knowledge of the world in order to decide what to do. Of course, you will have to give it that knowledge in a form it can use, and write the program that uses it. You’ll want to use the techniques from the second part of the course to do this.

There are three parts to this project:

1. You need to develop an agent that explores a Wumpus World, where the rules of the world are given in advance (i.e., they are as described in AIMA). This part is worth 40% of the total.

2. Next, you need to develop an agent that can accept a description of the rules of the world in Propositional Logic and then explore that world successfully. This part is also worth 40% of the total.

3. Finally, you need to develop an agent that can accept a description of the rules of the world in First-Order Logic and then explore that world successfully. This part is worth 20% of the total.

For each part of the project, 50% of the grade will be for the code and 50% for the writeup.

We will provide you with a Wumpus World simulator that you can run to test your agent. We will also use it to evaluate your agent.

Submit a zip or other archive of your work using Blackboard before the deadline. Include your writeup as a PDF in the archive. **Important:** Please name your archive starting with your last name (e.g., ferguson-project-02.zip). You would be amazed at how confusing it is to pull down 40 files named project.zip.
Part I: Wumpus Hunting Agent

I don’t think I have to say much about the basic Hunt the Wumpus setup. We have gone over it several times in class and the textbook has plenty of details and examples. There is one wumpus per game and it doesn’t move. Same with the gold. The percepts are exactly as described in the textbook. To win the game, your agent needs to “climb out” from the start location with the gold.

You can assume that the Wumpus World is always square, but the size of the grid should be a parameter to your agent. That is, we will run your agent in Wumpus Worlds of different (square) sizes. You can assume 4-connectivity as in the textbook. You cannot assume the locations of the wumpus, the gold, or of pits. These can be anywhere.

For this part of the project, you can assume that the world works exactly as described in the book. That is, you can code the reasoning about the world and decision-making however you think is best. I urge to read the descriptions of the other parts of the project before jumping in and cranking out code.

The point of this part is to get you comfortable with the infrastructure we will provide and with the idea of an “agent” program that perceives, reasons, and acts.

Part II: Propositional Worlds

This part of the project is just like the first part in terms of what your agent must do. However, instead of you knowing in advance what the rules of the world are, your agent will be given this as a machine-readable description in propositional logic. You then need to use this description to inform your agent’s decisions. That is, because the rules of the world are not given in advance, you cannot simply write program logic (“business rules”) to drive the agent.

The idea here is to appreciate why it makes sense to separate the business rules from the code that drives the agent. Although you will be reading in a description of the world, in general you might be learning your world knowledge from experience, or by being told, or whatever. If you were a company developing Wumpus Hunters that had to operate in a wide variety of environments, you would probably find that separating the description of world from the program code allows you develop variations on the program more quickly and with fewer bugs (of course, there can be bugs in the knowledge someone writes down, or learns for that matter).

If you can do this part, you can obviously do Part I also. Just code up the description of the AIMA Wumpus World in Propositional Logic and use it in the Part I program. But you don’t have to do Part I that way if you don’t want to. As with Part I, read the description of Part III before you jump in and start writing code.
Part III: First-Order Worlds

If you ace parts I and II of this project, you should extend your agent to accept a description of the world in First-Order Logic. As you will see, this knowledge will generally be considerably more concise, understandable, and debuggable than a propositional description. In fact, since a proposition is simply a 0-ary predicate (true or false with no arguments), this version of your agent would be able to handle either description.

The idea here is to understand what the expressiveness of first-order logic gives you, as well as what it costs you in terms of computational complexity and implementation effort. The agent probably won’t perform any better or worse (depending on your implementation), but it should be informative.

Other Ideas

It’s easy to come up with variations on the Wumpus World that could be fun to explore. For example, having several agents exploring a world would be easy. One issue that could arise in this case is that some actions would no longer be deterministic. For example, if two agents are in the room with the gold and they both try to grab it at the same time, only one will get it. For the other agent, the action will fail even though the agent’s knowledge says it should succeed (given that the agent is in the room with the gold).

It would also be easy to allow the wumpus (or the gold) to move. This time it would be the reasoning from perceptions that would become fallible. Absent any constraints on the behavior of these objects, you really wouldn’t be able to infer anything about them. However, it might be reasonable to know things such as the wumpus only moves in a small radius (say it doesn’t go far from home). Or it only moves some number of times. Or whatever. We don’t have the representational tools yet to handle these environments in any generality, but things could be tried.

Programming Practice (same as Project 1)

You can use any programming language you like. If you don’t have a preference, I recommend Java since the compiler will help you avoid many simple mistakes. Python is a reasonable option also. Lisp (or perhaps Clojure) would certainly be appropriate for this project if you know it or would like to get to know it.

Include a README.txt describing how to build your program. It would probably be a good idea to learn how to use make if you don’t already, but various IDEs like Eclipse are possible
also. The easier you make it for us to build and run your program, the better it is for everybody. If you aren’t familiar with these concepts, seek help early.

You should probably not need any non-standard libraries for this project. If you do, be very clear about the requirements in your writeup and in the README.txt. When in doubt, check with the TAs early.

Writing Up Your Work (almost the same as Project 1)

As noted above, it is crucial that you present your work clearly, honestly, and in its best light. We will give lots of credit for good writeups. We will not like submissions with sloppy, unstructured, hard to read writeups that were clearly written at the last minute.

Your goal is to produce a technical report of your efforts for an expert reader. You need to convince the reader that you knew what you were doing and that you did it as best you could in the (entire) time available.

Write up what you did (and why). Don’t write a narrative about how you developed your program (“first I edited this file, then I looked up this, then I learned this, …”). Instead, describe the problems you identified and what approach and techniques you used to address them. That is, describe your solutions (or attempted solutions, see below), not your workflow.

If you didn’t get to something in your code, write about what you might have done. (There’s always something you might have done.) Be specific. You need to convince us that you know what to do, not just that you know the keywords.

If something didn’t work, write about why and what you would do differently. Again, be specific. Also, don’t write “I would have started sooner.” Your readers are not idiots. If something doesn’t work, the TAs might be able to help you find the problem if you go to them early enough.

Provide references to relevant material that you used in your project. Of course you can’t use other people’s code, but if you find good ideas somewhere, you can code them yourself and then document the usage in your writeup.

Your report must be your own words. Material taken from other sources must be properly attributed if it is not digested and reformulated in your own words. Plagiarism is cheating.

Start your writeup early, at least in outline form. Document your design decisions as you make them. That way half the write-up is done by the time you’re running the program. Don’t forget to include illustrations of the program in action (traces, screenshots) and some kind of evaluation.

Your report must be typeset (not handwritten). Using \LaTeX and BibLaTeX makes things look nice.
and is worth learning anyway if you don’t know it, but it’s not a requirement. Your report must be submitted as a PDF file.

The length of the report is up to you. For a CSC242 assignment, I would say that 3–5 pages is the minimum that would allow you to convince the reader. Be sure to include screenshots and/or traces (which wouldn’t count towards the 3-5 pages minimum of actual text).