

# **Intelligent Systems Study Guide**

Don Colton  
Brigham Young University–Hawaii

March 19, 2012

# Preface

This book is a study guide for the CS 490R class (formerly CS 440) at Brigham Young University–Hawaii entitled “Intelligent Systems” or “Introduction to Artificial Intelligence” as taught by Professor Don Colton.

The content of the book is centered on three projects and two skills that will be main parts of the curriculum of this course. It also contains typical final exam questions.

The skills are (a) working with conditional probabilities, and (b) solving propositional logic systems by resolution.

The projects are (a) robotic vacuum cleaner, (b) Wumpus hunter, and (c) speech recognition of numbers.

Each skill or project is the subject of a chapter that provides background and training.

## Test Bank

As material is covered in the book, exam questions are inserted to show what the student should be learning. These exam questions appear throughout the book, together with answers. At the back of the book, Appendix [E](#) (page [82](#)) is a Test Bank. It repeats these same questions that appeared throughout the book, but without their answers.

The Test Bank is a way for students to test themselves by reviewing the questions and making sure they know at least one acceptable answer.

The Test Bank is also a way for teachers to be reminded of specific things that students should be able to answer.

Sometimes the questions and answers summarize material that is presented

nearby in greater detail.

Sometimes the questions and answers are the actual presentation of that material. This is especially true when the specific material is something simple like vocabulary, and repetition would be tiresome and redundant.

Following is the format in which questions and answers are presented.

**Exam Question 1** (p.82):

What does AI stand for?

**Acceptable Answer:**

artificial intelligence

The questions are linked to make it easy for the student to jump back and forth between the test bank and the content chapters.

# Contents

1	Readings	5
2	Conditional Probability	6
3	Bayesian Probability	12
4	Propositional Resolution	16
5	Projects in General	22
6	Vacuum	25
7	Hunt the Wumpus	29
8	Number Recognition	32
9	Exam Topics	39
A	CC2001: Intelligent Systems	46
B	Vacuum Driver Source Code	55
C	Wumpus Driver Source Code	66
D	Numbers Driver Source Code	78

<i>CONTENTS</i>	4
<b>E Test Bank</b>	<b>82</b>
<b>Index</b>	<b>88</b>

# Chapter 1

## Readings

You are assigned to read in the textbook for a certain amount of time each week. (See the syllabus for details.) However, your specific readings are self directed.

Our textbook, *Artificial Intelligence, A Modern Approach, Second Edition*, by: Stuart Russell and Peter Norvig (1080 pages. ISBN: 0-13-790395-2. Prentice Hall.) is a department rental. (The third edition has been published but we will continue to use the second edition because it is adequate for our needs.)

While reading, you should prepare notes (talking points) about your readings so you can report in class. Be prepared to talk for about three minutes over what you studied, why you selected it, what you learned, and where you think your studies might lead you next.

Chapter 9 (page 39) contains a preliminary list of topics that you should know about. Read through that list and if something catches your attention, make a note of it and follow up in your readings.

Appendix A (page 46) identifies learning objectives considered important by the curriculum committees established by the two largest professional societies in computing. Again, read through that list and if something catches your attention, make a note of it and follow up in your readings.

If you think you might like to study Artificial Intelligence / Intelligent Systems at graduate school, you are strongly encouraged to dig deeply into the textbook. It is excellent and will give you incredible preparation for that field.

## Chapter 2

# Conditional Probability

### Contents

---

<a href="#">2.1 Strategy</a> . . . . .	7
<a href="#">2.2 Notation</a> . . . . .	7
<a href="#">2.3 Sample Problems</a> . . . . .	8

---

Probability, Conditional Probability, and Updating of Probability are important skills relating to Artificial Intelligence and Intelligent Systems.

This is true because intelligent systems may be required to select between different possible actions where the outcomes are not certain, but can be estimated in terms of **probability**. To make the best choice, expected values must be calculated.

[http://en.wikipedia.org/wiki/Conditional\\_probability](http://en.wikipedia.org/wiki/Conditional_probability)

For this class, we will test you by giving you probability information and asking you to calculate other probability information.

Probabilities are always numbers between zero (meaning it never happens) and one (meaning it always happens). We will normally write them as fractions.

You are given certain probabilities. Your task is to find the requested probability. Express your answer as a reduced fraction (like 5/7).

## 2.1 Strategy

Most students are familiar with the **Venn diagram**.

[http://en.wikipedia.org/wiki/Venn\\_diagram](http://en.wikipedia.org/wiki/Venn_diagram) has a wonderful article that gives useful information. Even students familiar with Venn may learn something new.

The recommended approach to solving problems in conditional probability is to construct a Venn diagram using the facts at hand. Then, using the Venn diagram, determine the answer.

## 2.2 Notation

You should be familiar with the commonly used notations relating to probability. These are often written with mathematical symbols.

[http://en.wikipedia.org/wiki/Truth\\_table](http://en.wikipedia.org/wiki/Truth_table) has much more.

**Primitives:** These are some of the primitive wordings and operations used with probability.

$p(x)$  : When you see  $p()$  read it as “the probability that ... is true”.

$\cap$  : When you see  $\cap$  read it as the word “and”. It can also be read as the word “intersection”. It can be pronounced “cap”. It is also called **conjunction**.

$\cup$  : When you see  $\cup$  read it as the word “or”. It can also be read as the word “union”. It can be pronounced “cup”. It is also called **disjunction**.

$\bar{x}$  : When you see a bar over something, read it as the word “not”.

$|$  : When you see  $|$  read it as the word “given”.

$\rightarrow$  : When you see  $\rightarrow$  read it as the word “implies”. Note that “implies” is not the same as “causes”.

**Expressions:** These are typical combinations of the five primitives into longer expressions.

$p(A)$  means “the probability that A is true”.

$p(A)=5/7$  means that in the universe of possibilities, there are basically seven equally likely groupings of things, and in five of them A is true.

$p(A \cap B)$  means the (joint) probability that both A and B are true. We may

also write this as  $\mathbf{p(A\ and\ B)}$ .

$p(\overline{B})$  means “the probability that not B is true”, or in other words, “the probability that B is false”. We may also write this as  $\mathbf{p(not\ B)}$ .

$p(A\cap\overline{B})$  means the probability that A is true and B is false. We may also write this as  $\mathbf{p(A\ and\ not\ B)}$ .

$p(A|B)$  means the probability that A is true if we already know that B is true. We may also write this as  $\mathbf{p(A\ given\ B)}$ .

$p(A\rightarrow B)$  means the probability that if A is true, then B is also true. We may also write this as  $\mathbf{p(A\ implies\ B)}$ .

## 2.3 Sample Problems

<http://quizgen.doncolton.com/> quiz q45 provides additional opportunities for you to learn and practice these skills.

Given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A\cap B)=1/9$ .

You should first derive the following Venn diagram: (4(2)5)7.

The “(4(2)” part represents the A circle in the **Venn diagram**. It means that in four cases, A is true but B is not true. In two cases A is true and B is true. It does not say anything about when A is false.

The “(2)5)” part represents the B circle in the Venn diagram. It means that in two cases, B is true and A is also true. In five cases B is true but A is not true. It does not say anything about when B is false.

The “7” part represents the space outside the A and B circles. It means that in seven cases both A and B are false.

Each of these 4, 2, 5, and 7 cases are independent and equally likely, for a total of 18 possible cases.

### Exam Question 2 (p.82):

Find  $p(A\cap\overline{B})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A\cap B)=1/9$ .

### Acceptable Answer:

$2/9$

### Exam Question 3 (p.82):

Find  $p(\overline{A}\cap B)$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A\cap B)=1/9$ .

### Acceptable Answer:

5/18

**Exam Question 4 (p.82):**

Find  $p(\bar{A} \cap \bar{B})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .

**Acceptable Answer:**

7/18

**Exam Question 5 (p.82):**

Find  $p(A|B)$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .

**Acceptable Answer:**

2/7

**Exam Question 6 (p.82):**

Find  $p(A|\bar{B})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .

**Acceptable Answer:**

4/11

**Exam Question 7 (p.82):**

Find  $p(B|A)$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .

**Acceptable Answer:**

1/3

**Exam Question 8 (p.82):**

Find  $p(B|\bar{A})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .

**Acceptable Answer:**

5/12

Given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ , you should first derive the following Venn diagram: (3(7)1)1.

**Exam Question 9 (p.82):**

Find  $p(\bar{A} \cap B)$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .

**Acceptable Answer:**

1/12

**Exam Question 10 (p.82):**

Find  $p(\bar{A} \cap \bar{B})$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .

**Acceptable Answer:**

1/12

**Exam Question 11 (p.82):**

Find  $p(A|B)$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .

**Acceptable Answer:**

$$7/8$$

**Exam Question 12 (p.82):**

Find  $p(A|\bar{B})$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A\cap B)=7/12$ .

**Acceptable Answer:**

$$3/4$$

**Exam Question 13 (p.82):**

Find  $p(B|A)$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A\cap B)=7/12$ .

**Acceptable Answer:**

$$7/10$$

**Exam Question 14 (p.83):**

Find  $p(B|\bar{A})$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A\cap B)=7/12$ .

**Acceptable Answer:**

$$1/2$$

Given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ , you should first derive the following Venn diagram: (7(2)7)5.

**Exam Question 15 (p.83):**

Find  $p(A\cap\bar{B})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .

**Acceptable Answer:**

$$1/3$$

**Exam Question 16 (p.83):**

Find  $p(\bar{A}\cap B)$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .

**Acceptable Answer:**

$$1/3$$

**Exam Question 17 (p.83):**

Find  $p(\bar{A}\cap\bar{B})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .

**Acceptable Answer:**

$$5/21$$

**Exam Question 18 (p.83):**

Find  $p(A|B)$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .

**Acceptable Answer:**

$$2/9$$

**Exam Question 19 (p.83):**

Find  $p(A|\bar{B})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .

**Acceptable Answer:**

$7/12$

**Exam Question 20 (p.83):**

Find  $p(B|A)$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .

**Acceptable Answer:**

$2/9$

**Exam Question 21 (p.83):**

Find  $p(B|\bar{A})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .

**Acceptable Answer:**

$7/12$

## Chapter 3

# Bayesian Probability

### Contents

---

<b>3.1</b>	<b>The Product Rule</b>	<b>12</b>
<b>3.2</b>	<b>Detailed Example</b>	<b>13</b>
<b>3.3</b>	<b>Bayes' Rule</b>	<b>14</b>

---

[http://en.wikipedia.org/wiki/Thomas\\_Bayes](http://en.wikipedia.org/wiki/Thomas_Bayes) has information about Thomas Bayes (17021761).

Bayes comes up a lot in AI in connection with probabilistic inference. It is surprisingly simple.

### 3.1 The Product Rule

The **product rule** is this:

$$p(a \wedge b) = p(a|b)p(b) \text{ or } p(a \wedge b) = p(b|a)p(a)$$

We will show an example of this shortly.

Combining these two forms, we get **Bayes' rule**, aka Bayes' law, aka Bayes' theorem, which is this:

$$p(b|a) = p(a|b)p(b)/p(a)$$

(For some reason, I have a hard time remembering the Bayes' rule, but a somewhat easier time remembering the product rule. Fortunately for me, it is easy to derive Bayes from product.)

Each of  $p(a)$  and  $p(b)$  are called **prior probabilities**, or **a priori probabilities**. They are the probabilities that something is true when you don't know anything else about anything.

$p(a|b)$  is the conditional probability that  $a$  is true given that you already know  $b$  is true.

**Exam Question 22** (p.83):

What is the product rule?

**Acceptable Answer:**

$$p(a \text{ and } b) = p(a \text{ given } b)p(b)$$

**Exam Question 23** (p.83):

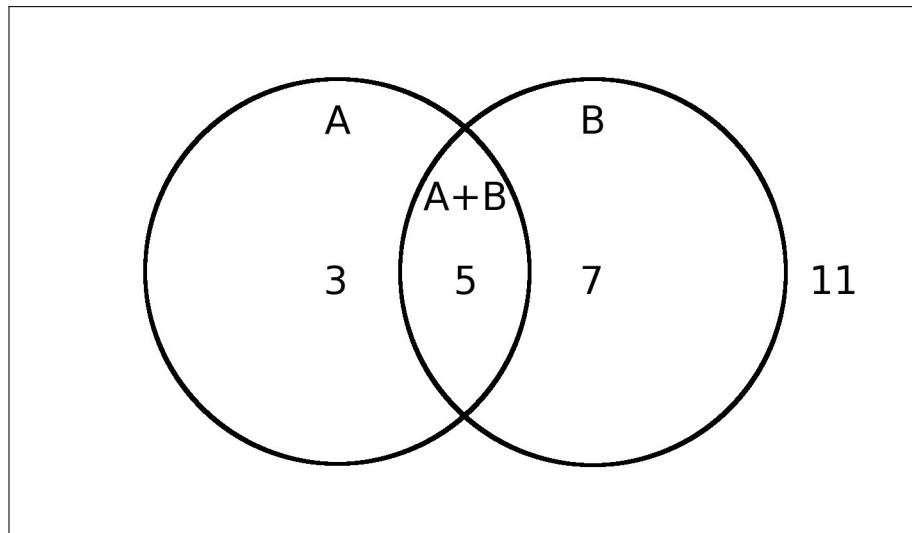
What is Bayes' rule?

**Acceptable Answer:**

$$p(a \text{ given } b) = p(b \text{ given } a)p(a)/p(b)$$

## 3.2 Detailed Example

Let's look at what this means by way of a Venn diagram.



In this example,  $p(A) = 8/26$ ,  $p(A \wedge B) = 5/26$ , and  $p(B) = 12/26$ .

Bayes' rule is used to calculate  $p(b|a)$  when our basic facts include  $p(a|b)$ ,  $p(a)$ , and  $p(b)$ . We do this as follows.

We know that  $p(A \wedge B) = p(A|B)p(B)$ . What does that mean?  $p(A|B)$  is the probability of  $A$  given that  $B$  is known to be true.

In this case, we have  $5+7=12$  cases where  $B$  is true, and in 5 of them,  $A$  is also true. Hence,  $p(A|B)$  is  $5/12$ .

We now have our three facts:  $p(A|B) = 5/12$ ,  $p(A) = 8/26$ , and  $p(B) = 12/26$ .

The product rule tells us that multiplying  $p(A|B)$  by  $p(B)$  we get  $p(A \wedge B)$ . Specifically,  $5/12$  times  $12/26$  equals  $5/26$ . Notice that the 12s cancel each other out.

The product rule tells us that dividing  $p(A \wedge B)$  by  $p(A)$  we get  $p(B|A)$  which is our goal.  $5/26$  divided by  $8/26$  equals  $5/8$ . Notice that the 26s cancel each other out.

We can check our work by counting up the chances directly. We see there are 8 chances for  $a$  to be true, and in 5 of them,  $b$  is also true. Therefore  $p(b|a)$  is  $5/8$ .

Bayes' rule just combines the two applications of the product rule.

$$p(B|A) = p(A|B)p(B)/p(A)$$

$$p(B|A) = (5/12)(12/26)/(8/26)$$

### 3.3 Bayes' Rule

Let's look at two propositions.

A: The sound ah was uttered.

B: The computer thinks it heard the sound ah.

We want to know the probability that ah was uttered when the computer recognizes an ah:  $p(u|r)$ .

We can find the prior probability that ah was **uttered** by looking at a corpus of speech labeled by trained human transcribers. Let's make up a number and say that out of some corpus of speech, ah is uttered in one percent of the frames.

$$p(u) = 0.01$$

We can find the prior probability that ah was **recognized** by looking at a similar corpus of speech labeled by computer. Let's make up a number and

say that in that corpus of speech, ah is recognized in two percent of the frames.

$$p(r) = 0.02$$

Due to confusion between similar phonemes, let's say that out of the times ah was actually uttered, it is recognized as the most likely phoneme 3/4 of the time.

$$p(r|u) = 0.75$$

From this we can calculate  $p(u|r)$ , the probability that ah was uttered given the computer recognized it.

$$p(u|r) = p(r|u)p(u)/p(r) = 0.75 * 0.01 / 0.02 = 0.375.$$

I guess in this case our computer is not very accurate yet.

# Chapter 4

## Propositional Resolution

### Contents

---

4.1	Notation . . . . .	17
4.2	How to Resolve . . . . .	18
4.3	Canonical Grading Form . . . . .	19
4.4	Sample Problems . . . . .	19

---

Resolution is the process of combining truth clauses to arrive at the most simplified version of those clauses.

We will restrict our attention to the domain called **Propositional Logic** or “Propositional Calculus”. (There is another more complicated domain called **Predicate Calculus** or “first order predicate calculus” or simply **first order logic**.) In the propositional domain, the basic (atomic) clauses are simple propositions that have a well-defined truth value, being either true or false.

For example, the clauses “It is raining” and “If it is raining then the ground will be wet” can be used to deduce that “The ground will be wet.”

Many years ago philosophers and logicians created lists of rules by which these deductions could be made. They are called **rules of inference**, or sometimes **sylogisms**. Many of them have Latin names, like **modus ponens**.

[http://en.wikipedia.org/wiki/Modus\\_ponens](http://en.wikipedia.org/wiki/Modus_ponens) has an introduction to the subject, and includes a list of other rules of inference in propositional calculus and predicate calculus.

In 1965 the philosopher / mathematician / computer scientist **John Alan Robinson** pointed out that all these rules of inference can be simplified down to a reliable and complete process called **resolution**.

[http://en.wikipedia.org/wiki/Resolution\\_\(logic\)](http://en.wikipedia.org/wiki/Resolution_(logic)) discusses resolution.

We expect you to learn and correctly apply resolution to a series of clauses that you will be given. We explain how to do that below.

## 4.1 Notation

**Vocabulary:** The following words are important.

**Conjunction** means “and” and is sometimes represented by the symbol  $\wedge$ .

**Disjunction** means “or” and is sometimes represented by the symbol  $\vee$ .

**Basics:** (1) The list of TRUE statements is called the knowledge base (the KB). We will add and delete statements to improve the list. (2) Each statement is called a clause, and consists of an or-list (**disjunction**) of simple propositions. If it is in the KB, it is claimed to be TRUE. (3) Each simple proposition is either TRUE or FALSE. Put another way, for every proposition (p), the clause “(p) or (not p)” must be TRUE.

“I am dry” is a simple proposition, and “I have an umbrella” is a simple proposition. Typically we abbreviate propositions down to letters or short words for convenience. The statement “I am dry, or I don’t have an umbrella” could be written as “(dry) or (not umbrella)” or “dry -umb” or even “d -u”. Each such statement is called a clause. As mentioned above, every clause in the KB is asserted to be TRUE.

We will express our knowledge base as a **conjunction** of **disjunctions**.

[http://en.wikipedia.org/wiki/Conjunctive\\_normal\\_form](http://en.wikipedia.org/wiki/Conjunctive_normal_form) discusses this form, conjunctive normal form, which is also called **CNF**.

Each clause is a **disjunction** of propositions. In a clause expressed as a **disjunction** of propositions, one of the propositions must be true. If one proposition is true, then the whole clause becomes true.

Each clause in the knowledge base must be true. When all must be true, we call that a **conjunction**.

## 4.2 How to Resolve

**Rule 1 (clause simplification):** We can drop any proposition we know to be FALSE from a clause because the whole clause must be TRUE, and the FALSE part clearly isn't helping. This is exactly like regular math where adding zero does not change anything. If  $x+0=5$  we can drop the  $+0$  and be left with  $x=5$ . If (a) or (b) or (c) is TRUE, and we know that (a) is FALSE, then we can drop it, leaving (b) or (c) is TRUE.

**Rule 2 (reduction):** Adding more propositions to a clause (an or-list) cannot change it from TRUE to FALSE. That is the nature of OR. If we know (b) is TRUE, then (b) or (c) is also TRUE, no matter what (c) is. As a direct result, if we have two clauses in the KB, and the little one is an exact subset of the big one, we can throw away the big one without losing any information. For example, if we know (i can drive), it does not help to also say (i can drive) or (i am rich). It tells nothing about my riches.

**Rule 2 (special case):** If we have a clause that includes “a -a” within it, we can delete it from the KB. The clause provides no information, since we already know that either (a) or (not a) is TRUE. It is called a **tautology**, and the “a” and “-a” are called **complementary literals**.

**Rule 3 (resolution):** New clauses can be created from old clauses. This part is a little tricky, but it is very powerful. If we have the clause “a x”, and we also have the clause “-a y”, we can combine them to create a new clause. Notice that one includes “a” in its or-list, and the other includes the opposite, “-a”. By basic rule 1, (a) is either TRUE or FALSE. Here is the tricky part. IF (a) is TRUE, then (not a) must be FALSE, so the second clause simplifies into (y) is TRUE. On the other hand, if (a) is FALSE, then the first clause simplifies into (x) is TRUE. We can splice the original clauses together to deduce “x y”. More generally, if we have (lots of things) or (a), and (other things) or (not a), we can combine them into (lots of things) or (other things). That's resolution.

**Rule 3 (gotcha):** If you have “a b more1” and “-a -b more2” there is a temptation to conclude “more1 more2”. However, one cannot combine on both “a” and “b” at the same time. The correct combination on “a” would result in “b more1 -b more2” which includes “b -b” which is always true. Hence by the special case already mentioned, the conclusion can be dropped.

**Procedure:** To simplify the KB, we look at pairs of clauses. If one is a subset of the other, we delete the bigger one. If they share complementary

propositions (one has (a) and the other has (not a)) we use resolution to generate a new clause and insert it into the KB (unless the new clause was already there). We continue looking at all pairs of clauses until no more insertions or deletions can be made. Careful ordering can make the job much faster; always reduce when you can. But whatever order you do things, you will always get the same result in the end.

### 4.3 Canonical Grading Form

There are many ways to write the same thing. The clauses could potentially be in any order. The propositions within the clauses could potentially be in any order.

For ease in grading, we require the following order. This makes all correct answers look the same, so grading can be done by exact match. (It sounds harder than it is.)

- (1) First arrange each clause into alphabetical order. The sorting order is - (dash) then [a-z].
- (2) Have a single space before and after each proposition. This makes it easier for us to catch mistakes before grading occurs.
- (3) Next order the whole clauses into alphabetical order.
- (4) Have a single space between clauses.

Everything should fit onto a single line, a **conjunction of disjunctions, conjunctive normal form (CNF)**.

Additional lines, such as comments, may not be included.

### 4.4 Sample Problems

Reduce each of the following to its most simple form through resolution. Each letter represents a proposition that is either TRUE or FALSE. Each (clause) is a TRUE statement (at least one proposition is TRUE).

<http://quizgen.doncolton.com/> quiz q41 provides additional opportunities for you to learn and practice these skills.

**Exam Question 24 (p.83):**

Resolve:  $(\neg a \ c) (\neg b \ a) (\neg c \ a) (\neg c \ b) (b \ c)$

**Acceptable Answer:**

( a ) ( b ) ( c )

**Exam Question 25 (p.83):**

Resolve: ( -a b ) ( -b c ) ( a b )

**Acceptable Answer:**

( b ) ( c )

**Exam Question 26 (p.83):**

Resolve: ( -c -d a ) ( a b d ) ( a c ) ( a c d )

**Acceptable Answer:**

( -d a ) ( a b ) ( a c )

**Exam Question 27 (p.83):**

Resolve: ( -b -c a ) ( -c b d ) ( -d a b )

**Acceptable Answer:**

( -c a ) ( -c b d ) ( -d a b )

**Exam Question 28 (p.83):**

Resolve: ( -a d ) ( -b c ) ( -c a ) ( -d a c ) ( a b )

**Acceptable Answer:**

( -b c ) ( a ) ( d )

**Exam Question 29 (p.83):**

Resolve: ( -b a ) ( -b a c ) ( -c -d a ) ( a b c )

**Acceptable Answer:**

( -b a ) ( -d a ) ( a c )

**Exam Question 30 (p.83):**

Resolve: ( -a -b c ) ( -b -c d ) ( -b a )

**Acceptable Answer:**

( -b a ) ( -b c ) ( -b d )

**Exam Question 31 (p.83):**

Resolve: ( -a -c b ) ( -a -d c ) ( -b d ) ( -c a b )

**Acceptable Answer:**

( -a -b c ) ( -a -d b ) ( -a -d c ) ( -b d ) ( -c b ) ( -c d )

**Exam Question 32 (p.83):**

Resolve: ( -c -d a ) ( -d b c ) ( a b ) ( a c d ) ( c d )

**Acceptable Answer:**

$$(-c -d a) (a b) (b c) (c d)$$

**Exam Question 33** (p.83):

$$\text{Resolve: } (-a -d c) (-d a) (a b)$$

**Acceptable Answer:**

$$(-d a) (-d c) (a b)$$

# Chapter 5

## Projects in General

### Contents

---

<b>5.1</b>	<b>How To Submit</b>	<b>22</b>
<b>5.2</b>	<b>Choice of Language</b>	<b>23</b>
<b>5.3</b>	<b>Bake Off</b>	<b>24</b>

---

A substantial portion of your learning in this class will come as a result of programs that you write.

### 5.1 How To Submit

You will submit your projects by copying the files into a special inbox on the IS2 machine. The location of the inbox is:

```
~dc/inbox/
```

The inbox is set up as a “write only” destination. That means you can copy things into the inbox but you cannot edit them or retrieve them from the inbox.

To submit your work, first, create or import the files into your own file space on the IS2 machine. Make sure the files are complete and work properly.

Next, copy (do not move) the files into the inbox. For example, if your file is named “John1”, you would copy it using the following command:

```
cp John1 ~dc/inbox/
```

Note: “mv” also works, kind of, but sometimes fails. “cp” is more reliable.

Note: Some students have attempted to upload their files directly into the inbox. Very often this fails because the inbox is “write only.” It is much more reliable to upload the files into your own file space first and then to copy them to the inbox.

Note: Some students have attempted to create their files directly within the inbox by using a text editor. Very often this fails because the inbox is “write only.” It is much more reliable to create the files into your own file space first and then to copy them to the inbox.

## 5.2 Choice of Language

You can use any programming language that supports these features:

- (a) Programs will be run under Linux on the IS2 machine.
- (b) Your program can use the name assigned by the instructor. Typically this consists of your own name followed by one or more digits. The first version of my program might be assigned the name “Don1” and the second version “Don2”.
- (c) Your program must successfully run from the command line by typing its assigned name, such as “./Don1”. There will be no command line arguments.
- (d) Your program can receive and process character input provided through STDIN.
- (e) Your program can provide character output through STDOUT.
- (f) Your program is permitted to have and to create additional files, but all additional files must start with the assigned name followed by a dot. If the name is “yadda” (for example), your main executable file must be named “yadda” and you may have additional files named “yadda.\*” where the “\*” can be anything, including a directory (folder) within which your files can be named anything you like.

C and C++ are known to work well.

Perl is known to work well.

Ruby is known to work well.

Tcl/Expect is known to work well.

Java is known to work, but it's a bit more tricky. You will probably use a “wrapper” for your primary executable, and it will start your actual program. Also, Java seems to have very long start-up times which can make your program appear to run slowly.

### 5.3 Bake Off

We will compare student programs (projects) by competing in a **bake off**. This terminology has reference to contests of cooking skills where each baker is required to produce some item of food that will be judged. By judging the food item we assess the skills of the baker.

Essentially we will give each program the same starting point and then run them simultaneously. Programs will earn points according to the rules of each project. Those with the most points will receive the better grades.

You will have access to the testing harness under which your program will be judged. This will allow you to test your program before submitting it for grading.

# Chapter 6

## Vacuum

### Contents

---

<b>6.1</b>	<b>Vacuum Rules</b>	.....	<b>25</b>
<b>6.2</b>	<b>Vacuum Driver</b>	.....	<b>27</b>
<b>6.3</b>	<b>Vacuum Agent</b>	.....	<b>27</b>

---

For this task, you will program a **vacuum** cleaner. You should visit every place in the room, clean what is dirty, return to your home base, and quit.

The room is divided into a grid of squares. Each square is either occupied by an obstacle, or is dirty, or is clean. One square is designated as your home square.

### 6.1 Vacuum Rules

Your program is a vacuum that cleans an arbitrary room of grid cells. Make your program an executable that will run from the command line.

Invoke the driver with a list of clients on the command line.

Example: `vacDriver vacIdiot vacIdiot vacIdiot`

(You can read the driver source code for more information.)

A **percept** is something perceived by the agent (the robot). At the start of each turn, you receive a vector of percepts that tell you all the things you can currently perceive. If this were an actual robot instead of a virtual one,

the percepts would be provided by physical hardware built into the robot.

All communication with your program is via STDIN and STDOUT. On each turn, your vacuum will receive via STDIN the current set of percepts. On each turn, your vacuum must respond within 1 second with an action. (Don't worry. 1 second is a long time. This is just to let the driver respond properly to failed agents that have dropped into an infinite loop.)

The room is generated at random, on a square-cell pattern, with walls and furnishings filling some of the cells. The vacuum starts at a randomized home location and must finish at that same location.

The driver prompts the vacuum by sending a set of percepts. The percepts are, in order, radar-left, radar-front, radar-right, dirt, and home. Each is a binary quantity, with 1 representing true and 0 representing false. The percepts are space-separated and terminated by newline. For example, "1 0 0 1 0\n".

The vacuum responds by giving a command. The valid commands are: "forward\n": move forward one cell, if possible, else do not move. "left\n": turn left 90 degrees, staying in the same cell. "right\n": turn right 90 degrees, staying in the same cell. "vacuum\n": pick up dirt in the current cell. "off\n": turn off, indicating the task is completed.

For the benefit of human players, the driver will reply with "what?" in case an invalid command is entered. Robotic vacuums should not need this functionality.

For the benefit of robotic players, the driver will echo and ignore any line starting with a "#" mark. This allows the robot to make reports to its programmer to aid in debugging.

The driver creates a visual display of each vacuum's activity, showing the starting random seed, a diagram of the room and vacuum, and a score to date. By using the same random seed, different vacuums can be compared. Score is -100 per dirt remaining, +100 per dirt captured, -1 per command issued, and +100 for ending in the proper cell (direction faced does not matter). Since it is anticipated that all vacuums will seek out all dirt and eliminate it, the score differences will be based on how quickly the task is accomplished.

Setup:

Download the vacDriver. chmod it to be executable. It is a tcl/expect program.

Download the `vacDriver`. `chmod` it to be executable. It is a Perl program.

On the command line, type the following command:

```
./vacDriver vacIdiot vacIdiot vacIdiot vacIdiot
```

This will run the driver with four copies of the “idiot” sample program. (We call it the idiot because it just moves randomly with no planning.)

Create your own program that will behave like the idiot, only lots smarter. Be prepared to submit it in class for competition with agents written by other students.

In the bake off, each vacuum will receive the same starting location and room configuration. All will run until a winner is determined. Total points earned will be compared to determine grades.

## 6.2 Vacuum Driver

Appendix B (page 55) presents a version of the driver program that will be used to evaluate student programs.

## 6.3 Vacuum Agent

Following is a sample agent program that is used to illustrate the basics of how the student program could be constructed.

```
#!/usr/bin/perl -w

$moves = 0;
while ( 1 ) {
    chomp ( $line = <STDIN> );
    # print "# $line\n";
    ( $rl, $rf, $rr, $d, $h ) = split ( / /, $line );
    $r = rand(); $moves++;
    if ( $d ) { print "vacuum\n"; next }
    if ( $moves > 1 && $h ) { print "off\n"; last }
    if ( $rf && $rl ) { print "right\n"; next }
    if ( $rf && $rr ) { print "left\n"; next }
    if ( $rl && $rr ) { print "forward\n"; next }
}
```

```
if ( $rf ) { $dir = "left";
  if ( $r > 0.5 ) { $dir = "right" }
  print "$dir\n"; next }
if ( $rl ) { $dir = "forward";
  if ( $r > 0.85 ) { $dir = "right" }
  print "$dir\n"; next }
if ( $rr ) { $dir = "forward";
  if ( $r > 0.85 ) { $dir = "left" }
  print "$dir\n"; next }
$dir = "forward";
if ( $r > 0.95 ) { $dir = "left" }
if ( $r < 0.05 ) { $dir = "right" }
print "$dir\n";
}
```

## Chapter 7

# Hunt the Wumpus

### Contents

---

<a href="#">7.1 Wumpus Rules</a>	<a href="#">30</a>
<a href="#">7.2 Wumpus Driver</a>	<a href="#">31</a>
<a href="#">7.3 Wumpus Agent</a>	<a href="#">31</a>

---

Your task in the Wumpus World is to explore a small system of caves, retrieve the gold if possible, avoid death at the hands of the **Wumpus**, and avoid death by falling into one of the pits.

The **Wumpus** is a mythical creature that does not like adventurers.

Your task is to earn points by searching for Gold in a cave system. The system is a four-by-four grid. You begin in the lower left corner (1,1) and you are facing right (East).

You are guaranteed that your initial position and the two squares you can reach are safe from hazards.

You should decide your moves by calculating the probability of various dangers. You initially know that the Wumpus is located in one of the other 13 cells. You also know that the gold is located in one of the 13 cells. You also know that the 13 cells each have a 20% chance of being a pit.

## 7.1 Wumpus Rules

Maximize your points, which you gain and lose as follows. Cost: -1 for each action Cost: -10 for shooting arrow Penalty: -10 for bumping into a wall Penalty: -1000 for death (by pit or Wumpus) Reward: +1000 for getting the gold Reward: +100 for each new room visited / explored Reward: +100 for quitting back at the entrance (1,1) Reward: +100 for killing the Wumpus

Make your program an executable that will run from the command line.

Invoke the driver with a list of clients on the command line.

Example: `./wDriver wIdiot wIdiot wIdiot`

(You can read the driver source code for more information.)

All communication with your program is via STDIN and STDOUT. On each turn, your hunter will receive via STDIN as set of percepts. On each turn, your hunter must respond within 1 second with an action. (Don't worry. 1 second is a long time.)

The percepts come as a comma-separated list of five bits of information: (stench,breeze,glitter,bump,scream) Example: `n,n,n,n,n` Stench means you can smell the Wumpus in an adjacent cell Breeze means you can tell there is a pit in an adjacent cell Glitter means there is gold in the current cell. You win. Game over. Bump means you tried to move forward but ran into a wall and did not move. Scream means you shot the Wumpus and it is now dead and no longer dangerous.

You must reply one of (S,shoot,L,left,R,right,A,forward,G,grab,Q,quit)

Setup:

Download the wDriver. `chmod` it to be executable. It is a tcl/expect program.

Download the wIdiot. `chmod` it to be executable. It is a Perl program.

On the command line, type the following command:

```
./wDriver wIdiot wIdiot wIdiot wIdiot
```

This will run the driver with four copies of the "idiot" sample program. (We call it the idiot because it just moves randomly with no planning.)

Create your own program that will behave like the idiot, only lots smarter. Be prepared to submit it in class for competition with agents written by

other students.

## 7.2 Wumpus Driver

Appendix C (page 66) presents a version of the driver program that will be used to evaluate student programs.

## 7.3 Wumpus Agent

Following is a sample agent program that is used to illustrate the basics of how the student program could be constructed.

```
#!/usr/bin/perl -w

$moves = 0;
while ( 1 ) {
    chomp ( $line = <STDIN> );
    # print "# $line\n";
    ( $stench, $breeze, $glitter, $bump, $scream ) =
        split ( /,/, $line );
    $action = "forward";
    $r = int ( rand ( 4 ) ); $moves++;
    if ( $r == 0 ) { $action = "left" }
    if ( $r == 1 ) { $action = "right" }
    print "$action\n";
}
```

# Chapter 8

## Number Recognition

### Contents

---

8.1	Overview	32
8.2	Preparation	33
8.3	Phonemes	33
8.4	Vocabulary	35
8.5	Over Training	35
8.6	Evaluation	36
8.7	Testing Your Program	37
8.8	Numbers Driver	38

---

### 8.1 Overview

Your task is to write a program that converts **phonemes** into **numbers**. The phonemes are presented as a time-aligned transcription where each phoneme has a starting time, an ending time, and a symbolic representation. The resulting numbers are to be expressed in written characters as whole individual words.

## 8.2 Preparation

CSLU is the Center for Spoken Language Understanding. I am familiar with it because it was the home to my PhD program back in the 1990s.

We will use as our standard the CSLU Numbers corpus. A sample of this corpus is available for free download from the CSLU Corpus website. At this writing, the homepage for CSLU corpora is:

<http://www.cslu.ogi.edu/corpora/corpCurrent.html>

Find the Numbers Corpus, probably version 1.3.

Download the sample file, probably called `numbers_sample.zip` and store it somewhere so you can use it for the remainder of this project.

Open the file and review its contents. You should find a folder (directory) named “labels” and another folder named “trans”. The labels directory contains the time-aligned phonetic labels in files each with a `.phn` extension. The trans directory contains the orthographic (normal writing) transcriptions in files each with a `.txt` extension.

All files are in plain ASCII text.

Display: Your instructor may ask you to display a matching pair of files, one `.phn` and one `.txt`, to show that you understand the file organization and to verify that you can work with the files.

## 8.3 Phonemes

Phonemes are sounds from which utterances are built up. Commonly phonemes can be divided into two categories: vowels and consonants.

The word “cat” generally is spoken using the three phonemes “k”, “a”, “t”.

The word “Kate” generally is spoken using the four phonemes “k”, “eh”, “ee”, “t”.

### **IPA, Worldbet, and OGIbet English Broad Phonetic Labels**

There are around 50 different phonemes in English. You can find them described on the chart on the next page.

# IPA, Worldbet, and OGIbet English Broad Phonetic Labels

*Center for Spoken Language Understanding – Oregon Graduate Institute of Science & Technology*

IPA	Worldbet	OGIbet	Example	Category
i:	i:	iy	be <u>e</u> t	Front Vowels
I	I	ih	bi <u>i</u> t	
ε	E	eh	be <u>e</u> t	
æ	@	ae	ba <u>a</u> t	
ɪ	I_x	ix	rose <u>s</u>	Central Vowels  (British)
ʊ	u_x	ux	sui <u>t</u>	
ə	&	ax	ab <u>o</u> ve	
ə	&0		to go <u>o</u>	
ɒ	5		po <u>t</u>	
u	u	uw	bo <u>o</u> t	Back Vowels
ʊ	U	uh	bo <u>o</u> k	
ʌ	^	ah	ab <u>o</u> ve	
ɔ	>	ao	ca <u>u</u> ght	
ɑ	A	aa	fa <u>t</u> her	
ɝ	3r	er	bi <u>r</u> d	Retroflexes
ɝ	&r	axr	butte <u>r</u>	
ei	ei	ey	ba <u>y</u>	Diphthongs  (British) (British) (British)
aI	aI	ay	by <u>e</u>	
ɔi	>i	oy	bo <u>y</u>	
iʊ	iU		fe <u>w</u>	
aʊ	aU	aw	abo <u>u</u> t	
oʊ	oU	ow	bo <u>o</u> t	
iə	i&		he <u>r</u> e	
eə	e&		the <u>r</u> e	
uə	u&		po <u>o</u> r	
p <sup>h</sup>	ph	p	pa <u>n</u>	Voiceless Plosives
t <sup>h</sup>	th	t	ta <u>n</u>	
k <sup>h</sup>	kh	k	ca <u>n</u>	
b	b	b	ba <u>n</u>	Voiced Plosives
d	d	d	da <u>n</u>	
g	g	g	gande <u>r</u>	
m	m	m	me <u>e</u>	Nasals
n	n	n	kn <u>ee</u>	
ŋ	N	ng	sing <u>e</u>	
r <sub>l</sub>	th_ (	dx	wri <u>t</u> er	Flaps
r <sub>d</sub>	d_ (	dx	rid <u>e</u> r	
f	f	f	fi <u>n</u> e	Voiceless Fricatives
θ	T	th	thi <u>g</u> h	
s	s	s	si <u>g</u> n	
ʃ	S	sh	assu <u>r</u> e	
h	h	hh	ho <u>p</u> e	
v	v	v	vi <u>n</u> e	Voiced Fricatives
ð	D	dh	thi <u>g</u>	
z	z	z	resi <u>g</u> n	
ʒ	Z	zh	azu <u>r</u> e	
tʃ	tS	ch	ch <u>u</u> rch	Affricates
dʒ	dZ	jh	judg <u>e</u>	
l	l	l	le <u>n</u> t	Glides  (approximants)
ɹ	9r	r	re <u>n</u> t	
j	j	y	ye <u>s</u>	
w	w	w	w <u>e</u> nt	
m	m=	em	botto <u>m</u>	Syllabics
n	n=	en	butto <u>n</u>	
ŋ	N=	eng		
l	l=	el	bottl <u>e</u>	

IPA	Worldbet	OGIbet	Example	Category
	pc	pcl	_pa <u>n</u>	Voiceless
	tc	tc1	_ta <u>n</u>	Plosive
	kc	kcl	_ca <u>n</u>	Closures
	bc	bcl	_ba <u>n</u>	Voiced
	dc	dcl	_da <u>n</u>	Plosive
	gc	gcl	_ga <u>n</u> der	Closures
	tSc	chcl	_ch <u>u</u> rch	Affricate
	dZc	jhcl	_judg <u>e</u>	Closures
	+	.epi	epi <u>n</u> thetic closure	

IPA	Worldbet	OGIbet	Type of Diacritic
t <sup>h</sup>	_h	-h	aspirated
	_x		centralized
t̥ d̥	_l		dental
	_ (		flapped (consonant)
	_F		fricated stop
	_?*	q	glottal onset
ʔ	_?	-q	glottalized
d <sup>l</sup>	_l		lateral release
i:	_:	-el	lengthened
d <sup>n</sup>	_n		nasal release
ẽ	_~	-n	nasalized
	_NL	.nitl	not in the language
t <sup>j</sup>	_j		palatalized
ɝ	_r	-r	retroflexion
ɶ	_i		less rounded
ɶ	_w		more rounded
ɹ	_=		syllabicity
ɹ	_v		voiced
n̥ d̥	_0		voiceless
	_*	-	waveform cut off

<i>Worldbet, as modified at OGI</i>			
	_fp	-fp	filled pause
	_ln	-ln	line noise corruption
	_bn		background noise

Worldbet	OGIbet	Non Speech Sound Item
.bn	.bn	background noise
.br	.br	breath noise
.cough	.cough	cough
.ct	.ct	clear throat
.laugh	.laugh	laugh
.ln	.ln	lin noise
.ls	.ls	lip smack
.ns	.ns	human, not speech
.sneeze	.sneeze	sneeze
.tc	.tc	tongue click

<i>Worldbet, as modified at OGI</i>		
.beep	.beep	beep
.burp	.burp	burp
.fp	.fp	filled pause
.pau	.pau	pause or silence
.sniff	.sniff	sniff
.uu	.unk	unintelligible speech
.vs	.vs	squeak, voice crack
.glot	glot	glottalization

## 8.4 Vocabulary

The full CSLU Numbers corpus consists of about 23,900 utterances, each with a phonetic transcription and an orthographic transcription. The transcriptions were prepared by trained human transcribers.

The sample corpus consists of 283 transcribed utterances, or about one percent of the full corpus.

The following (64 or so) words will be counted in the scoring.

a and double eight eighteen eighteenth eighth eightieth eighty eleven eleventh fifteen fifteenth fifth fiftieth fifty first five fortieth forty four fourteen fourteenth fourth half hundred hundredth nine nineteen nineteenth ninetieth ninety ninth oh one second seven seventeen seventeenth seventh seventieth seventy six sixteen sixteenth sixth sixtieth sixty ten tenth third thirteen thirteenth thirtieth thirty thousand three triple twelfth twelve twentieth twenty two zero

All other words and fragments of words will be ignored.

However, you may need to account for the fact that some non-scored words will occur in the utterances that you are processing.

## 8.5 Over Training

Your program is performing a recognition task. There are two ways to approach such a task. One is by strict memorization. You check to see which of the 283 cases you are viewing and you report the correct results for that case.

The other is by a careful combination of memorization (**basis** cases) and rules (**induction**). Rather than memorizing the 283 sample results, you may memorize the 46 words that occur in the sample results. And you may create suitable rules to combine the memorized results to make additional, non-memorized results.

A brief example may help. When we calculate  $4+3$ , and we are very small, we may hold up four fingers on one hand and three on the other hand. Then we may count all the fingers to arrive at a total of seven. In this example, the meaning of 4 was memorized, the meaning of 3 was memorized, and the process of combining them was memorized. 4 and 3 are part of the basis.

The process is part of the induction.

Later in life, we may actually memorize our “addition facts” or “addition tables” and come to know that  $4+3=7$  without ever counting it out. We can tackle a problem like  $639+15$  by breaking it up into smaller problems:  $9+5=14$ ; write down the 4 and carry the 1.  $3+1+1$  is 5; write it down and carry zero. 6 comes down for a total of 654. In this process, the single-digit additions were memorized and the rules for carry and column order provide the inductive step to we can add numbers of any size.

Because this same situation occurs in recognizing numbers, we must be careful to keep our basis set as small as possible, within reason. It is good to divide your corpus (the 283 utterances) into parts. One part can be used for development. You can look at those utterances in great detail and analyze them to understand your task. Another part can be used for testing. You use those utterances to see how well your program performs. The great risk is that you will train your program to recognize your 283 utterances very well, but you will fail to recognize the other 23,600 utterances. That is called **over training**. Use your resources carefully.

## 8.6 Evaluation

I intend to give you more than the 283 utterances that are in the sample set. If so, you must not disclose them beyond this class. That is part of the license agreement I signed in order to get the corpus. You can study the additional utterances to improve your performance.

When you believe you have a reasonable performance rate for your program, you can request me to test it. I will test a maximum of five versions of your program. Your score for the assignment will be the score you earn on your fifth (or last) test.

When I test your program, I will report the number of utterances it correctly recognized, meaning that the entire utterance was exactly right in every way. I will provide you with my testing script, but not with the data files that I am using.

The input files (.phn) are given to you in their original form, redirected to you as STDIN standard input, with results captured from you as STDOUT standard out. Your file should present one word per line, interspersed at your discretion with comment lines that I will ignore. Comment lines will

be identified by a “#” as the first character of the line.

Each corpus transcription file (.txt) will be “normalized” by being stripped of all words that start with dot (such as .pau), and all characters that are enclosed in angle brackets (such as <bn>). Letters will be converted to lower case. Runs of white space will be converted to a single space. The 28 characters a-z, apostrophe, and space, will be kept. All other characters will be deleted.

Most but not all human transcriptions are correct. A few are wrong, so it is impossible to get a perfect score. Also some transcriptions include word fragments. But try to get the best score you can.

## 8.7 Testing Your Program

Here is a sample program you can key in and test.

```
#!/usr/bin/perl --
while ( $in = <STDIN> ) { print "# got input: $in" }
print "one two three four five\n" ;# your answer
```

You can test it by following these steps.

- (1) Create a directory for this project in your account on the is2 machine.
- (2) Inside that directory, create these symbolic links:

```
ln -s ~dc/cs440/numbers/data
ln -s ~dc/cs440/numbers/nDriver
```

- (3) Put your program in your directory. We will assume you have called it “Don0”.
- (4) Test your program by running the following command line.

```
./nDriver ./Don0 data/sample/NU-1*
```

- (5) You should see two program executions, with details of your program’s operation.

The summary tells your accuracy, (number correct / number tested), and your score,  $(-1000 * \log_2(\text{error rate}))$ , max 9999.

Every time you cut your error rate in half, your score goes up by 1000.

An accuracy of 0.00 (none right) results in a score of 0.

An accuracy of 0.50 results in a score of 1000.

An accuracy of 0.75 results in a score of 2000.

An accuracy of 0.875 results in a score of 3000.

The maximum score is 9999.

Driving down your error rate really drives up your score.

(6) Now improve your program to print the correct answer each time.

(7) You can submit your program to the instructor by copying it as follows:

```
cp Don0 ~/dc/inbox/
```

## 8.8 Numbers Driver

Appendix D (page 78) presents a version of the driver program that will be used to evaluate student programs.

## Chapter 9

# Exam Topics

On the final exam you will be asked to discuss important topics from the textbook or our class discussions. You will write from memory about the subject, telling what it is, why it is important, and other comments you may wish to share. Students typically write for two to three minutes on each topic.

Following is a preliminary list of topics based on prior semesters readings and discussions.

In addition to this list, consider the learning objectives mentioned in chapter [A](#) (page 46).

**Exam Question 34** (p.83): Discuss: 1.0 Artificial Intelligence

**Exam Question 35** (p.83): Discuss: 1.1 Turing test

**Exam Question 36** (p.83): Discuss: 1.1 automated reasoning

**Exam Question 37** (p.83): Discuss: 1.1 machine learning

**Exam Question 38** (p.84): Discuss: 1.1 total Turing test

**Exam Question 39** (p.84): Discuss: 1.1 agent

**Exam Question 40** (p.84): Discuss: 1.2 dualism

**Exam Question 41** (p.84): Discuss: 1.2 materialism

**Exam Question 42** (p.84): Discuss: 1.2 logical positivism

**Exam Question 43** (p.84): Discuss: 1.2 algorithm

**Exam Question 44** (p.84): Discuss: 1.2 Godel's incompleteness theorem

**Exam Question 45** (p.84): Discuss: 1.2 intractability

**Exam Question 46** (p.84): Discuss: 1.2 NP-completeness

**Exam Question 47** (p.84): Discuss: 1.3 machine evolution

**Exam Question 48** (p.84): Discuss: 1.3 genetic algorithms

**Exam Question 49** (p.84): Discuss: 1.3 expert systems

**Exam Question 50** (p.84): Discuss: 1.3 frames

**Exam Question 51** (p.84): Discuss: 2.2 rational agent

**Exam Question 52** (p.84): Discuss: 2.2 autonomous agent

**Exam Question 53** (p.84): Discuss: 2.3 simple reflex agent

**Exam Question 54** (p.84): Discuss: 2.3 goal-based agent

**Exam Question 55** (p.84): Discuss: 2.3 utility-based agent

**Exam Question 56** (p.84): Discuss: 2.4 accessible environment

**Exam Question 57** (p.84): Discuss: 2.4 deterministic environment

**Exam Question 58** (p.84): Discuss: 2.4 episodic environment

**Exam Question 59** (p.84): Discuss: 2.4 static vs dynamic environment

**Exam Question 60** (p.84): Discuss: 2.4 discrete vs continuous environment

**Exam Question 61** (p.84): Discuss: 3.x search

**Exam Question 62** (p.85): Discuss: 3.x path cost

**Exam Question 63** (p.85): Discuss: 3.x breadth-first search

**Exam Question 64** (p.85): Discuss: 3.x uniform-cost search

**Exam Question 65** (p.85): Discuss: 3.x depth-first search

**Exam Question 66** (p.85): Discuss: 3.x depth-limited search

**Exam Question 67** (p.85): Discuss: 3.x iterated deepening search

**Exam Question 68** (p.85): Discuss: 3.x bidirectional search

**Exam Question 69** (p.85): Discuss: 4.x heuristics

**Exam Question 70** (p.85): Discuss: 4.x best-first search

**Exam Question 71** (p.85): Discuss: 4.x greedy search

**Exam Question 72** (p.85): Discuss: 4.x A\* search

**Exam Question 73** (p.85): Discuss: 4.x iterated improvement

**Exam Question 74** (p.85): Discuss: 6.x knowledge representation

**Exam Question 75** (p.85): Discuss: 6.x inference (sound, complete)

**Exam Question 76** (p.85): Discuss: 6.x propositional logic

**Exam Question 77** (p.85): Discuss: 7.x first-order logic

**Exam Question 78** (p.85): Discuss: 7.x atomic sentence

**Exam Question 79** (p.85): Discuss: 7.x predicate

**Exam Question 80** (p.85): Discuss: 7.x quantified sentence

**Exam Question 81** (p.85): Discuss: 7.x situational calculus

**Exam Question 82** (p.85): Discuss: 7.x diagnostic rules

**Exam Question 83** (p.85): Discuss: 7.x causal rules

**Exam Question 84** (p.85): Discuss: 9.x unification

**Exam Question 85** (p.85): Discuss: 9.x Modus Ponens

**Exam Question 86** (p.86): Discuss: 9.x Horn form

**Exam Question 87** (p.86): Discuss: 9.x resolution

**Exam Question 88** (p.86): Discuss: 9.x conjunctive normal form

**Exam Question 89** (p.86): Discuss: 9.x implicative normal form

**Exam Question 90** (p.86): Discuss: 13.x conditional plans

**Exam Question 91** (p.86): Discuss: 13.x execution monitoring

**Exam Question 92** (p.86): Discuss: 13.x action monitoring

**Exam Question 93** (p.86): Discuss: 13.x replanning agent

**Exam Question 94** (p.86): Discuss: 14.x prior probabilities

**Exam Question 95** (p.86): Discuss: 14.x conditional probabilities

**Exam Question 96** (p.86): Discuss: 14.x joint probability distribution

**Exam Question 97** (p.86): Discuss: 14.x Bayes' rule

**Exam Question 98** (p.86): Discuss: 14.x conditional independence

**Exam Question 99** (p.86): Discuss: 14.x Bayesian updating

**Exam Question 100** (p.86): Discuss: 15.x belief networks

**Exam Question 101** (p.86): Discuss: 15.x stochastic simulation

**Exam Question 102** (p.86): Discuss: 15.x truth-functional system

**Exam Question 103** (p.86): Discuss: 18.x performance element

**Exam Question 104** (p.86): Discuss: 18.x learning element

**Exam Question 105** (p.86): Discuss: 18.x inductive learning

**Exam Question 106** (p.86): Discuss: 19.x neural network

**Exam Question 107** (p.86): Discuss: 19.x perceptron

**Exam Question 108** (p.86): Discuss: 19.x linearly separable function

**Exam Question 109** (p.86): Discuss: 19.x feed-forward network

**Exam Question 110** (p.87): Discuss: 19.x back-propagation

**Exam Question 111** (p.87): Discuss: 19.x Bayesian learning

**Exam Question 112** (p.87): Discuss: 19.x multi-layer feed-forward network

**Exam Question 113** (p.87): Discuss: 22.x speech act

**Exam Question 114** (p.87): Discuss: 22.x phrase-structure grammar

**Exam Question 115** (p.87): Discuss: 22.x context-free grammar

**Exam Question 116** (p.87): Discuss: 22.x encoded message

**Exam Question 117** (p.87): Discuss: 22.x situated language

**Exam Question 118** (p.87): Discuss: 22.x augmented grammar

**Exam Question 119** (p.87): Discuss: 22.x pragmatic interpretation

**Exam Question 120** (p.87): Discuss: 22.x disambiguation

**Exam Question 121** (p.87): Discuss: 24.x image formation

**Exam Question 122** (p.87): Discuss: 24.x image processing

**Exam Question 123** (p.87): Discuss: 27.1 anytime algorithm

**Exam Question 124** (p.87): Discuss: 27.2 bounded optimality

**Exam Question 125** (p.87): Discuss: 27.2 prisoner's dilemma

## Appendix A

# CC2001: Intelligent Systems

The following eight pages are taken from CC2001: Computing Curricula 2001, Computer Science. It represents the 2001 view of computer science by the curriculum committees of **ACM**: the Association for Computing Machinery, and **IEEE-CS**: the Computer Society of the Institute for Electrical and Electronic Engineers.

In this class, we will be mostly interested in the following learning objectives.

**IS1:** 1, 2, 3, 5.

**IS2:** 1, 2, 5.

**IS3:** 1, 2, 3, 4.

**IS6:** 1, 2, 4, 6.

**IS7:** 3.

These would be good topics to investigate in your weekly readings.



# Computing Curricula 2001

## Computer Science

— Final Report —  
(December 15, 2001)

The Joint Task Force on Computing Curricula  
IEEE Computer Society  
Association for Computing Machinery

This material is based upon work supported by the  
National Science Foundation under Grant No. 0003263

Figure 5-1. Computer science body of knowledge with core topics underlined

<p><b>DS. Discrete Structures (43 core hours)</b> <u>DS1. Functions, relations, and sets</u> (6) <u>DS2. Basic logic</u> (10) DS3. Proof techniques (12) <u>DS4. Basics of counting</u> (5) <u>DS5. Graphs and trees</u> (4) <u>DS6. Discrete probability</u> (6)</p> <p><b>PF. Programming Fundamentals (38 core hours)</b> <u>PF1. Fundamental programming constructs</u> (9) <u>PF2. Algorithms and problem-solving</u> (6) <u>PF3. Fundamental data structures</u> (14) <u>PF4. Recursion</u> (5) <u>PF5. Event-driven programming</u> (4)</p> <p><b>AL. Algorithms and Complexity (31 core hours)</b> <u>AL1. Basic algorithmic analysis</u> (4) <u>AL2. Algorithmic strategies</u> (6) <u>AL3. Fundamental computing algorithms</u> (12) <u>AL4. Distributed algorithms</u> (3) <u>AL5. Basic computability</u> (6) AL6. The complexity classes P and NP AL7. Automata theory AL8. Advanced algorithmic analysis AL9. Cryptographic algorithms AL10. Geometric algorithms AL11. Parallel algorithms</p> <p><b>AR. Architecture and Organization (36 core hours)</b> <u>AR1. Digital logic and digital systems</u> (6) <u>AR2. Machine level representation of data</u> (3) <u>AR3. Assembly level machine organization</u> (9) <u>AR4. Memory system organization and architecture</u> (5) <u>AR5. Interfacing and communication</u> (3) <u>AR6. Functional organization</u> (7) <u>AR7. Multiprocessing and alternative architectures</u> (3) AR8. Performance enhancements AR9. Architecture for networks and distributed systems</p> <p><b>OS. Operating Systems (18 core hours)</b> <u>OS1. Overview of operating systems</u> (2) <u>OS2. Operating system principles</u> (2) <u>OS3. Concurrency</u> (6) <u>OS4. Scheduling and dispatch</u> (3) <u>OS5. Memory management</u> (5) OS6. Device management OS7. Security and protection OS8. File systems OS9. Real-time and embedded systems OS10. Fault tolerance OS11. System performance evaluation OS12. Scripting</p> <p><b>NC. Net-Centric Computing (15 core hours)</b> <u>NC1. Introduction to net-centric computing</u> (2) <u>NC2. Communication and networking</u> (7) <u>NC3. Network security</u> (3) <u>NC4. The web as an example of client-server computing</u> (3) NC5. Building web applications NC6. Network management NC7. Compression and decompression NC8. Multimedia data technologies NC9. Wireless and mobile computing</p> <p><b>PL. Programming Languages (21 core hours)</b> <u>PL1. Overview of programming languages</u> (2) <u>PL2. Virtual machines</u> (1) <u>PL3. Introduction to language translation</u> (2) <u>PL4. Declarations and types</u> (3) <u>PL5. Abstraction mechanisms</u> (3) <u>PL6. Object-oriented programming</u> (10) PL7. Functional programming PL8. Language translation systems PL9. Type systems PL10. Programming language semantics PL11. Programming language design</p>	<p><b>HC. Human-Computer Interaction (8 core hours)</b> <u>HC1. Foundations of human-computer interaction</u> (6) <u>HC2. Building a simple graphical user interface</u> (2) HC3. Human-centered software evaluation HC4. Human-centered software development HC5. Graphical user-interface design HC6. Graphical user-interface programming HC7. HCI aspects of multimedia systems HC8. HCI aspects of collaboration and communication</p> <p><b>GV. Graphics and Visual Computing (3 core hours)</b> <u>GV1. Fundamental techniques in graphics</u> (2) <u>GV2. Graphic systems</u> (1) GV3. Graphic communication GV4. Geometric modeling GV5. Basic rendering GV6. Advanced rendering GV7. Advanced techniques GV8. Computer animation GV9. Visualization GV10. Virtual reality GV11. Computer vision</p> <p><b>IS. Intelligent Systems (10 core hours)</b> <u>IS1. Fundamental issues in intelligent systems</u> (1) <u>IS2. Search and constraint satisfaction</u> (5) <u>IS3. Knowledge representation and reasoning</u> (4) IS4. Advanced search IS5. Advanced knowledge representation and reasoning IS6. Agents IS7. Natural language processing IS8. Machine learning and neural networks IS9. AI planning systems IS10. Robotics</p> <p><b>IM. Information Management (10 core hours)</b> <u>IM1. Information models and systems</u> (3) <u>IM2. Database systems</u> (3) <u>IM3. Data modeling</u> (4) IM4. Relational databases IM5. Database query languages IM6. Relational database design IM7. Transaction processing IM8. Distributed databases IM9. Physical database design IM10. Data mining IM11. Information storage and retrieval IM12. Hypertext and hypermedia IM13. Multimedia information and systems IM14. Digital libraries</p> <p><b>SP. Social and Professional Issues (16 core hours)</b> <u>SP1. History of computing</u> (1) <u>SP2. Social context of computing</u> (3) <u>SP3. Methods and tools of analysis</u> (2) <u>SP4. Professional and ethical responsibilities</u> (3) <u>SP5. Risks and liabilities of computer-based systems</u> (2) <u>SP6. Intellectual property</u> (3) <u>SP7. Privacy and civil liberties</u> (2) SP8. Computer crime SP9. Economic issues in computing SP10. Philosophical frameworks</p> <p><b>SE. Software Engineering (31 core hours)</b> <u>SE1. Software design</u> (8) <u>SE2. Using APIs</u> (5) <u>SE3. Software tools and environments</u> (3) <u>SE4. Software processes</u> (2) <u>SE5. Software requirements and specifications</u> (4) <u>SE6. Software validation</u> (3) <u>SE7. Software evolution</u> (3) <u>SE8. Software project management</u> (3) SE9. Component-based computing SE10. Formal methods SE11. Software reliability SE12. Specialized systems development</p> <p><b>CN. Computational Science (no core hours)</b> CN1. Numerical analysis CN2. Operations research CN3. Modeling and simulation CN4. High-performance computing</p>
---	---

Note: The numbers in parentheses represent the minimum number of hours required to cover this material in a lecture format. It is always appropriate to include more.

## Intelligent Systems (IS)

- IS1. Fundamental issues in intelligent systems [core]**
- IS2. Search and constraint satisfaction [core]**
- IS3. Knowledge representation and reasoning [core]**
- IS4. Advanced search [elective]**
- IS5. Advanced knowledge representation and reasoning [elective]**
- IS6. Agents [elective]**
- IS7. Natural language processing [elective]**
- IS8. Machine learning and neural networks [elective]**
- IS9. AI planning systems [elective]**
- IS10. Robotics [elective]**

The field of artificial intelligence (AI) is concerned with the design and analysis of autonomous agents. These are software systems and/or physical machines, with sensors and actuators, embodied for example within a robot or an autonomous spacecraft. An intelligent system has to perceive its environment, to act rationally towards its assigned tasks, to interact with other agents and with human beings.

These capabilities are covered by topics such as computer vision, planning and acting, robotics, multiagents systems, speech recognition, and natural language understanding. They rely on a broad set of general and specialized knowledge representations and reasoning mechanisms, on problem solving and search algorithms, and on machine learning techniques.

Furthermore, artificial intelligence provides a set of tools for solving problems that are difficult or impractical to solve with other methods. These include heuristic search and planning algorithms, formalisms for knowledge representation and reasoning, machine learning techniques, and methods applicable to sensing and action problems such as speech and language understanding, computer vision, and robotics, among others. The student needs to be able to determine when an AI approach is appropriate for a given problem, and to be able to select and implement a suitable AI method.

### **IS1. Fundamental issues in intelligent systems [core]**

*Minimum core coverage time:* 1 hour

#### *Topics:*

- History of artificial intelligence
- Philosophical questions
  - The Turing test
  - Searle’s “Chinese Room” thought experiment
  - Ethical issues in AI
- Fundamental definitions
  - Optimal vs. human-like reasoning
  - Optimal vs. human-like behavior
- Philosophical questions
- Modeling the world
- The role of heuristics

#### *Learning objectives:*

1. Describe the Turing test and the “Chinese Room” thought experiment.
2. Differentiate the concepts of optimal reasoning and human-like reasoning.
3. Differentiate the concepts of optimal behavior and human-like behavior.

4. List examples of intelligent systems that depend on models of the world.
5. Describe the role of heuristics and the need for tradeoffs between optimality and efficiency.

## **IS2. Search and constraint satisfaction [core]**

*Minimum core coverage time: 5 hours*

*Topics:*

Problem spaces  
Brute-force search (breadth-first, depth-first, depth-first with iterative deepening)  
Best-first search (generic best-first, Dijkstra's algorithm, A\*, admissibility of A\*)  
Two-player games (minimax search, alpha-beta pruning)  
Constraint satisfaction (backtracking and local search methods)

*Learning objectives:*

1. Formulate an efficient problem space for a problem expressed in English by expressing that problem space in terms of states, operators, an initial state, and a description of a goal state.
2. Describe the problem of combinatorial explosion and its consequences.
3. Select an appropriate brute-force search algorithm for a problem, implement it, and characterize its time and space complexities.
4. Select an appropriate heuristic search algorithm for a problem and implement it by designing the necessary heuristic evaluation function.
5. Describe under what conditions heuristic algorithms guarantee optimal solution.
6. Implement minimax search with alpha-beta pruning for some two-player game.
7. Formulate a problem specified in English as a constraint-satisfaction problem and implement it using a chronological backtracking algorithm.

## **IS3. Knowledge representation and reasoning [core]**

*Minimum core coverage time: 4 hours*

*Topics:*

Review of propositional and predicate logic  
Resolution and theorem proving  
Nonmonotonic inference  
Probabilistic reasoning  
Bayes theorem

*Learning objectives:*

1. Explain the operation of the resolution technique for theorem proving.
2. Explain the distinction between monotonic and nonmonotonic inference.
3. Discuss the advantages and shortcomings of probabilistic reasoning.
4. Apply Bayes theorem to determine conditional probabilities.

#### **IS4. Advanced search [elective]**

*Topics:*

- Genetic algorithms
- Simulated annealing
- Local search

*Learning objectives:*

1. Explain what genetic algorithms are and contrast their effectiveness with the classic problem-solving and search techniques.
2. Explain how simulated annealing can be used to reduce search complexity and contrast its operation with classic search techniques.
3. Apply local search techniques to a classic domain.

#### **IS5. Advanced knowledge representation and reasoning [elective]**

*Topics:*

- Structured representation
  - Frames and objects
  - Description logics
  - Inheritance systems
- Nonmonotonic reasoning
  - Nonclassical logics
  - Default reasoning
  - Belief revision
  - Preference logics
  - Integration of knowledge sources
  - Aggregation of conflicting belief
- Reasoning on action and change
  - Situation calculus
  - Event calculus
  - Ramification problems
- Temporal and spatial reasoning
- Uncertainty
  - Probabilistic reasoning
  - Bayesian nets
  - Fuzzy sets and possibility theory
  - Decision theory
- Knowledge representation for diagnosis, qualitative representation

*Learning objectives:*

1. Compare and contrast the most common models used for structured knowledge representation, highlighting their strengths and weaknesses.
2. Characterize the components of nonmonotonic reasoning and its usefulness as a representational mechanisms for belief systems.
3. Apply situation and event calculus to problems of action and change.
4. Articulate the distinction between temporal and spatial reasoning, explaining how they interrelate.
5. Describe and contrast the basic techniques for representing uncertainty.
6. Describe and contrast the basic techniques for diagnosis and qualitative representation.

## **IS6. Agents [elective]**

### *Topics:*

- Definition of agents
- Successful applications and state-of-the-art agent-based systems
- Agent architectures
  - Simple reactive agents
  - Reactive planners
  - Layered architectures
  - Example architectures and applications
- Agent theory
  - Commitments
  - Intentions
  - Decision-theoretic agents
  - Markov decision processes (MDP)
- Software agents, personal assistants, and information access
  - Collaborative agents
  - Information-gathering agents
- Believable agents (synthetic characters, modeling emotions in agents)
- Learning agents
- Multi-agent systems
  - Economically inspired multi-agent systems
  - Collaborating agents
  - Agent teams
  - Agent modeling
  - Multi-agent learning
- Introduction to robotic agents
- Mobile agents

### *Learning objectives:*

1. Explain how an agent differs from other categories of intelligent systems.
2. Characterize and contrast the standard agent architectures.
3. Describe the applications of agent theory, to domains such as software agents, personal assistants, and believable agents.
4. Describe the distinction between agents that learn and those that don't.
5. Demonstrate using appropriate examples how multi-agent systems support agent interaction.
6. Describe and contrast robotic and mobile agents.

## **IS7. Natural language processing [elective]**

### *Topics:*

- Deterministic and stochastic grammars
- Parsing algorithms
- Corpus-based methods
- Information retrieval
- Language translation
- Speech recognition

### *Learning objectives:*

1. Define and contrast deterministic and stochastic grammars, providing examples to show the adequacy of each.

2. Identify the classic parsing algorithms for parsing natural language.
3. Defend the need for an established corpus.
4. Give examples of catalog and look up procedures in a corpus-based approach.
5. Articulate the distinction between techniques for information retrieval, language translation, and speech recognition.

### **IS8. Machine learning and neural networks [elective]**

#### *Topics:*

- Definition and examples of machine learning
- Supervised learning
- Learning decision trees
- Learning neural networks
- Learning belief networks
- The nearest neighbor algorithm
- Learning theory
- The problem of overfitting
- Unsupervised learning
- Reinforcement learning

#### *Learning objectives:*

1. Explain the differences among the three main styles of learning: supervised, reinforcement, and unsupervised.
2. Implement simple algorithms for supervised learning, reinforcement learning, and unsupervised learning.
3. Determine which of the three learning styles is appropriate to a particular problem domain.
4. Compare and contrast each of the following techniques, providing examples of when each strategy is superior: decision trees, neural networks, and belief networks..
5. Implement a simple learning system using decision trees, neural networks and/or belief networks, as appropriate.
6. Characterize the state of the art in learning theory, including its achievements and its shortcomings.
7. Explain the nearest neighbor algorithm and its place within learning theory.
8. Explain the problem of overfitting, along with techniques for detecting and managing the problem.

### **IS9. AI planning systems [elective]**

#### *Topics:*

- Definition and examples of planning systems
- Planning as search
- Operator-based planning
- Propositional planning
- Extending planning systems (case-based, learning, and probabilistic systems)
- Static world planning systems
- Planning and execution
- Planning and robotics

*Learning objectives:*

1. Define the concept of a planning system.
2. Explain how planning systems differ from classical search techniques.
3. Articulate the differences between planning as search, operator-based planning, and propositional planning, providing examples of domains where each is most applicable.
4. Define and provide examples for each of the following techniques: case-based, learning, and probabilistic planning.
5. Compare and contrast static world planning systems with those need dynamic execution.
6. Explain the impact of dynamic planning on robotics.

**IS10. Robotics [elective]**

*Topics:*

Overview

- State-of-the-art robot systems
- Planning vs. reactive control
- Uncertainty in control
- Sensing
- World models

Configuration space

Planning

Sensing

Robot programming

Navigation and control

*Learning objectives:*

1. Outline the potential and limitations of today's state-of-the-art robot systems.
2. Implement configuration space algorithms for a 2D robot and complex polygons.
3. Implement simple motion planning algorithms.
4. Explain the uncertainties associated with sensors and how to deal with those uncertainties.
5. Design a simple control architecture.
6. Describe various strategies for navigation in unknown environments, including the strengths and shortcomings of each.
7. Describe various strategies for navigation with the aid of landmarks, including the strengths and shortcomings of each.

## Appendix B

# Vacuum Driver Source Code

Following is a version of the driver program that will be used to evaluate student programs.

```
#!/usr/bin/expect --
proc v args { return 0 } ;# verbosity low
proc v args { return 1 } ;# verbosity high
proc showAll ids { uplevel { showAllN $ids } } ;# three-up
proc showAll ids { uplevel { showAll1 $ids } } ;# one-up
proc showAll ids { uplevel { showAll2 $ids } } ;# two-up
proc showAll ids { uplevel { showAll3 $ids } } ;# three-up

# Vacuum Driver, for robotic vacuums cleaning arbitrary rooms
# written by Don Colton
# argv lists the executable agent programs (non-numeric names)
# argv may (but need not) contain a numeric random number seed
# we spawn each agent and communicate through stdin/stdout

# The room is generated at random, on a square-cell pattern,
# with walls and furnishings filling some of the cells. The
# vacuum starts at a randomized location and must finish at
# that same location.

# The driver prompts the vacuum by sending a set of percepts.
# The percepts are, in order, radar-left, radar-front, radar-
# right, dirt, and home. Each is a binary quantity, with 1
```

```

# representing true and 0 representing false. The percepts are
# space-separated and terminated by newline.
# For example, "1 0 0 1 0\n".

# The vacuum responds by giving a command. Valid commands are:
# "forward\n": move forward one cell, if possible.
# "left\n": turn left 90 degrees, staying in the same cell.
# "right\n": turn right 90 degrees, staying in the same cell.
# "vacuum\n": pick up dirt in the current cell.
# "off\n": turn off, indicating the task is completed.

# For the benefit of human players, the driver will reply with
# "what?" in case an invalid command is entered. Robotic
# vacuums should not need this functionality.

# For the benefit of robot players, the driver will echo and
# ignore any line starting with a "#" mark. This allows the
# robot to make reports to its programmer.

# The driver creates a visual display of the vacuum's working,
# showing the starting random seed, a diagram of the room and
# vacuum, and a score to date. By using the same random seed,
# different vacuums can be compared. Score is -100 per dirt
# remaining, +100 per dirt captured, -1 per command issued,
# and +100 for ending in the proper cell (direction faced does
# not matter). Since it is anticipated that all vacuums will
# seek out all dirt and eliminate it, the score differences
# will be based on how quickly the task is accomplished.

#-----
proc rand m {
    set device /dev/urandom          ;# /dev/random can block
    set fileId [open $device r]
    binary scan [read $fileId 4] i1 number
    set clipped [expr $number % $m]
    close $fileId
    return $clipped }
proc randomSeed {seed} { global RNDseed version;
    set RNDseed $seed; set version $seed; }
proc random15 {} { global RNDseed; # 15 bit int: 0..32767

```

```

    set RNDseed [expr $RNDseed * 1103515245 + 12345]
    expr int ( $RNDseed / 65536 ) % 32768 }
proc random {low high} {
    expr int( ($low)+[random15] * (($high)-($low)+1) / 32768) }
proc pick args {
    if { [llength $args] == 1 } { set args [lindex $args 0] }
    lindex $args [random 0 [expr [llength $args] - 1]] }
proc permute args { set out "";
    if { [llength $args] == 1 } { set args [lindex $args 0] }
    while { [llength $args] > 0 } {
        set nexti [random 0 [expr [llength $args] - 1]];
        lappend out [lindex $args $nexti];
        set args [lreplace $args $nexti $nexti];
    }; return $out; }
proc do {n body} { # based on p.123 of Tcl book
    global errorInfo errorCode
    while { $n > 0 } { incr n -1
        set code [catch { uplevel $body } string]
        if { $code == 1 } {
            return -code error -errorinfo $errorInfo -errorcode $errorCode $string }
        if { $code == 2 } { return -code return $string }
        if { $code == 3 } { return }; # break
        if { $code > 4 } { return -code $code $string }
    } }
#-----
# initialize a rectangle
proc blanket {roomIn rMin rMax cMin cMax color} {
    upvar $roomIn room
    for { set row $rMin } { $row <= $rMax } { incr row } {
        for { set col $cMin } { $col <= $cMax } { incr col } {
            set room($row,$col) $color } } }
#-----
proc genRoom roomIn { # generate a room
    upvar $roomIn room
    set room(score) 0
    set room(moves) 0
    set room(rMin) [set rMin 1]
    set room(rMax) [set rMax [random 12 18]]
    set room(cMin) [set cMin 1]
    set room(cMax) [set cMax [random 12 18]]

```

```

blanket room $rMin $rMax $cMin $cMax "#"
# =====
blanket room [expr $rMin + 1] [expr $rMax - 1] [expr $cMin + 1] [expr $cMax - 1] "

do [pick 3 4 4 5] { # generate some furniture
    set rW [pick 1 2 2 2 3 3]
    set r0 [random $rMin+1 $rMax-$rW]; set r9 [expr $r0+$rW-1]
    set cW [pick 1 2 2 2 3 3]
    set c0 [random $cMin+1 $cMax-$cW]; set c9 [expr $c0+$cW-1]
    blanket room $r0 $r9 $c0 $c9 "#" }

# position the vacuum somewhere
set room(vacR) [set vacR [random [expr $rMin+2] [expr $rMax-2]]]
set room(vacC) [set vacC [random [expr $cMin+2] [expr $cMax-2]]]
set room(homR) $vacR
set room(homC) $vacC
blanket room [expr $vacR-1] [expr $vacR+1] [expr $vacC-1] [expr $vacC+1] " "
set room(vacD) [pick N S E W]

# spread dirt around the room
for { set row $rMin } { $row < $rMax } { incr row } {
    for { set col $cMin } { $col < $cMax } { incr col } {
        if { $room($row,$col) == " " && [random 0 9] == 0 } {
            set room($row,$col) "d" } } }
set room(dirt) "dirty"
set room(moves) 0
set room(name) "initial"
}
#-----
# room layout is measured from the upper left corner
# (this is for ease of printing)
# 11 12 13 14 15 ... 1(cMax)
# 21 22 23 24 ...
# 11 is the NW corner
proc show roomIn { # show a room
    upvar map$roomIn room
    set output ""
    set dirty 0
    set score $room(score); # use a copy
    set rMin $room(rMin); # integer

```

```

set rMax $room(rMax); # integer
set cMin $room(cMin); # integer
set cMax $room(cMax); # integer
set vacR $room(vacR); # integer
set vacC $room(vacC); # integer
set homR $room(homR); # integer
set homC $room(homC); # integer
set vacD $room(vacD); # N S E W
for { set row $rMin } { $row <= $rMax } { incr row } {
  set out ""
  for { set col $cMin } { $col <= $cMax } { incr col } {
    if ![info exists room($row,$col)] {
      set room($row,$col) "#" }
    set cell $room($row,$col)
    if { $cell == "d" } { incr score -100; set dirty 1 }
    if { "$row,$col" == "$homR,$homC" } { set cell "o" }
    if { "$row,$col" == "$vacR,$vacC" } {
      if { $vacD == "N" } { set cell "A" }
      if { $vacD == "S" } { set cell "V" }
      if { $vacD == "E" } { set cell ">" }
      if { $vacD == "W" } { set cell "<" }
    }
    append out " $cell"
  }
  append output "$out\n"
}
# append output " score:$score\n"
if { $dirty == 0 } { set room(dirt) "clean" }
set room(okay) "failure"; set success ""
if { "$dirty$vacR,$vacC" == "0$homR,$homC" } {
  set room(okay) "success"; set success " SUCCESS!" }
append output " $roomIn s$room(score) m$room(moves) $room(name)"
if { $room(dead) == 1 } { append output " DEAD" }
return $output }
#-----
# this assumes all block lines are the same length
proc showAllN ids { # wide, all on one line
  uplevel { set output ""
    foreach id $ids { set output [merge $output [show $id]] }
    return $output } }

```

```

proc showAll1 ids {
    uplevel { set outputAll ""
        foreach id1 $ids { # one up
            set outputLine [show $id1]
            lappend outputAll $outputLine }
        join $outputAll "\n" } }
proc showAll2 ids {
    uplevel { set outputAll ""
        foreach {id1 id2} $ids { # three up
            set outputLine [show $id1]
            if { $id2 != "" } {
                set outputLine [merge $outputLine [show $id2]] }
            lappend outputAll $outputLine }
        join $outputAll "\n" } }
proc showAll3 ids {
    uplevel { set outputAll ""
        foreach {id1 id2 id3} $ids { # three up
            set outputLine [show $id1]
            if { $id2 != "" } {
                set outputLine [merge $outputLine [show $id2]] }
            if { $id3 != "" } {
                set outputLine [merge $outputLine [show $id3]] }
            lappend outputAll $outputLine }
        join $outputAll "\n" } }
proc merge {block1 block2} {
    set output ""
    set max 0; foreach line1 [split $block1 "\n"] {
        set len [string length $line1]
        if { $len > $max } { set max $len } }
    foreach line1 [split $block1 "\n"] line2 [split $block2 "\n"] {
        lappend output "[format %-${max}s $line1] $line2" }
    join $output "\n" }
#-----
proc move mapID {
    set roomIn "map$mapID"
    # send_user "$mapID: starting move ($roomIn)\n"
    upvar $roomIn room

    # compute and write the percept vector
    # room layout is measured from the upper left corner

```

```

# 11 12 13 14 15 ... 1(cMax)
# 21 22 23 24 ...
# 11 is the NW corner
set vacR $room(vacR)
set vacC $room(vacC)
set atX $room($vacR,$vacC)
set atE "#"; catch { set atE $room($vacR,[expr $vacC+1]) }
set atN "#"; catch { set atN $room([expr $vacR-1],$vacC) }
set atS "#"; catch { set atS $room([expr $vacR+1],$vacC) }
set atW "#"; catch { set atW $room($vacR,[expr $vacC-1]) }
set vacD $room(vacD)
if { $vacD == "E" } { set atF $atE; set atL $atN; set atR $atS }
if { $vacD == "N" } { set atF $atN; set atL $atW; set atR $atE }
if { $vacD == "S" } { set atF $atS; set atL $atE; set atR $atW }
if { $vacD == "W" } { set atF $atW; set atL $atS; set atR $atN }
if { $atF == "#" } { set atF 1 } else { set atF 0 }
if { $atL == "#" } { set atL 1 } else { set atL 0 }
if { $atR == "#" } { set atR 1 } else { set atR 0 }
if { $atX == "d" } { set atX 1 } else { set atX 0 }
if { "$room(homR),$room(homC)" == "$vacR,$vacC" } {
    set home 1 } else { set home 0 }
set percept "$atL $atF $atR $atX $home"

if [v] { send_user "$mapID: sending ($percept)\n" }
send "$percept\r"
expect "$percept\r\n"
# should be an overall timeout of 1 second incl comments
while { 1 } {
    set expect_out(1,string) "off" ;# timeout default command
    expect -re {[\r\n]*([\r\n]+)[\r\n]+}
    set cmd [string trim $expect_out(1,string)]
    # send_user "got ($cmd)\n"
    if ![regexp {^#} $cmd] break
    if [v] { send_user "$mapID: $cmd\n" } } ;# show comment
# send_user "broke with ($cmd)\n"

# accept the response and update the map
incr room(moves)
incr room(score) -1
if { $cmd == "forward" } {

```

```

    if { $room($vacR,$vacC) == " " } {
        set room($vacR,$vacC) "." }; # mark progress
    if { $atF == 1 } return
    if { $vacD == "N" } { incr room(vacR) -1 }
    if { $vacD == "S" } { incr room(vacR) +1 }
    if { $vacD == "E" } { incr room(vacC) +1 }
    if { $vacD == "W" } { incr room(vacC) -1 }
    return
}
if { $cmd == "left" } {
    if { $vacD == "E" } { set room(vacD) "N" }
    if { $vacD == "N" } { set room(vacD) "W" }
    if { $vacD == "S" } { set room(vacD) "E" }
    if { $vacD == "W" } { set room(vacD) "S" }
    return
}
if { $cmd == "right" } {
    if { $vacD == "E" } { set room(vacD) "S" }
    if { $vacD == "N" } { set room(vacD) "E" }
    if { $vacD == "S" } { set room(vacD) "W" }
    if { $vacD == "W" } { set room(vacD) "N" }
    return
}
if { $cmd == "vacuum" } {
    if { $room($vacR,$vacC) == "d" } {
        incr room(score) 100; set room($vacR,$vacC) " " }
    return
}
if { $cmd == "off" } {
    if { "$room(vacR),$room(vacC)" == "$room(homR),$room(homC)" } {
        incr room(score) 100 }
    return "dead" }
send_user "?? got ($cmd)\n"
return "dead" ;# declare the broken agent to be dead
# send "what?\r"
# expect "what?\r\n"
# return
}
#-----
# copy one array to create another

```

```

proc clone {from0 to0} { upvar $from0 from; upvar $to0 to
  foreach ele [array names from] { set to($ele) $from($ele) } }
#-----
randomSeed [exec date +%s]
# randomSeed 20020501

log_user 0
set timeout 1

if { $argc == 0 } {
  send_user "Usage: vacd \[seed] agent1 agent2 agent3 ... \n"
  send_user "  \[seed] is an optional random number seed. \n"
  send_user "  each agentN is the name of a program. \n"
  exit }

set playerCount 0
foreach agent $argv {
  if [regexp {[1-9][0-9]*} $agent] {
    randomSeed $agent; continue }
  incr playerCount
}

# send_user "There are $playerCount players \n"
send_user "Vacuum Driver for Robots, game # $version \n"

genRoom map
set map(dead) 0
# send_user "[show ""] \n"; # initial map

set n 0; set ids ""
foreach agent $argv {
  if [regexp {[1-9][0-9]*} $agent] continue
  lappend ids [incr n]
  # send_user "starting agent $n ($agent) \n"
  set agent2 $agent
  if { [file tail $agent2] == $agent2 } {
    set agent2 ".$agent2" }
  spawn $agent2
  set id2spawn($n) $spawn_id
  clone map map$n
}

```

```

    set map${n}(name) $agent }

set deads ""
set count 0
set moves 0
# set results ""

proc mygets args {
    global expect_out
    set expect_out(1,string) ""
    set timeout -1
    expect_user -re "(.*)\n"
    set ans $expect_out(1,string)
    return $ans }

send_user "[showAll $ids]\n"

while { 1 } {
    if { [incr count -1] < 1 } {
        if { $moves > 0 } {
            send_user "[showAll $ids]\n"
            # send_user "stopped agents: $deads\n"
        }
        send_user "press ENTER to continue, q to quit, num to fast-forward\n"
        set ans [mygets]
        if [regexp {[1-9][0-9]*$} $ans] { set count $ans }
        if [regexp {[Qq]} $ans] break
    }
    incr moves; set alive 0; set deads ""
    foreach id $ids {
        if { [set map${id}(dead)] == 1 } {
            lappend deads "$id"; continue }
        set alive 1
        set spawn_id $id2spawn($id)
        if { [move $id] == "dead" } {
            # send_user "[show $id]\n"
            set map${id}(dead) 1
            set m [format %3d [set map${id}(moves)]]
            set s [format %4d [set map${id}(score)]]
            set c [set map${id}(okay)]
        }
    }
}

```

```
        set n [set map${id}(name)]
        # lappend results "$id score=$s moves=$m name=$n $c"
    }
}
if { $alive == 0 } break
}

send_user "Final Standings\n"
send_user "[showAll $ids]\n"
# send_user "\nresults:\n[join [lsort $results] "\n"]\n"
send_user "Done (game # $version)\n"
```

## Appendix C

# Wumpus Driver Source Code

Following is a version of the driver program that will be used to evaluate student programs.

```
#!/usr/bin/expect --
fconfigure stdin -blocking 1

#####
# wumpus driver
puts "Welcome to Wumpus Driver by Don Colton"

#####
proc usage args { puts ""
    puts "usage 1: wumpDriver wumpClient wumpClient wumpClient ..."
    puts "  shows the map and allows direct competition"
    puts "usage 2: wumpDriver n wumpClient"
    puts "  runs one client n times and just reports the results."
    puts "optionally include seed=nnnn to seed the random generator."
    puts "optionally include width=n to print n across, default is 3."
    puts ""; exit }

#####
# agent gets percept: (stench,breeze,glitter,bump,scream)
# . example: n,n,n,n,n (comma separated list of five items)
# . stench means you can smell the wumpus in an adjacent cell
# . breeze means you can tell there is a pit in an adjacent cell
```

```

# . glitter means there is uncollected gold in the current cell
# . bump means you tried to move forward but ran into a wall and did not move
# . scream means you shot the wumpus and it is now dead and no longer dangerous
# agent must reply with one of (S,shoot,L,left,R,right,A,forward,Q,quit,G,grab)

set sc(Mv)    -1 ;# penalty -1 for each action
set sc(Bu)    -10 ;# penalty -10 for bumping into the wall
set sc(Sh)    -10 ;# penalty -10 for shooting arrow
set sc(Di)   -1000 ;# penalty -1000 for death
set sc(Go)    1000 ;# reward +1000 for getting the gold
set sc(RV)    100 ;# reward +100 for each new room visited / explored
set sc(QE)    100 ;# reward +100 for quitting back at the entrance (1,1)
set sc(KW)    100 ;# reward +100 for killing the wumpus

#####
# map() contains all game details in the following form
# map($agent,$xy) is defined where $agent has visited
# map($agent,x) is the x (col) where the agent is, [1-4], initially 1
# map($agent,y) is the y (row) where the agent is, [1-4], initially 1
# map($agent,dir) is the direction agent is facing: [^<v>X] (x if dead)
# map($agent,killed) is defined if wumpus is dead
# map($agent,perc) contains the current percepts
# map($agent,score) contains the score for the agent
# map($agent,shot) is defined if arrow has been shot
# map(b,$xy) is defined where there is a breeze (pit near)
# map($agent,g,$xy) is defined where there is gold
# map(p,$xy) is defined where there is a pit
# map(s,$xy) is defined where there is a stench (wumpus near)
# map(w,$xy) is defined where there is a wumpus

#####
# extract the seed if any
set seed ""; if [regexp {seed=(\d+)} $argv foo seed] {
  regsub " seed=$seed " " $argv " " " argv
  set argv [string trim $argv] }

# extract the width if any, default to 3
set width 3; if [regexp {width=(\d+)} $argv foo width] {
  regsub " width=$width" " $argv " " " argv
  set argv [string trim $argv] }

```

```
#####
# give names of the agents on the command line
set iter 1; set verbose 1
# puts "argv is ($argv)"
if { [llength $argv] == 2 && [regexp {^(\d+) (.*)} $argv foo iter argv] } {
    set verbose 0; puts "Running in Evaluation mode for $iter iterations." }
set count 0; foreach agent $argv {
    if ![regexp "/" $agent] { set agent "./$agent" }
    if ![file exists $agent] { puts "skipping $agent"; continue }
    incr count; lappend agents "a$count"
    set map(a$count,name) $agent; set map(a$count,pname) [file tail $agent] }
proc putsv {lvl line} { global verbose; if { $verbose >= $lvl } { puts $line } }
if { $count == 0 } { usage }
proc putsv1 {lvl line} { putsv $lvl $line }

#####
# draw the map showing the board configuration and score:
# each cell lists agent, wumpus, pit, gold (awpg)
# first dot is agent: [^v<>. ]; next wumpus [Ww ]; next pit [P ]; last gold [G ]
#####
# fred says: A
# +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
# |...|   |   |   |   |   |   |   |   |   |   |   |   |
# +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
# |   |   |   |   |   |   |   |   |   |   |   |   |
# +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
# |   |   |   |   |   |   |   |   |   |   |   |   |
# +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
# |   |   |   |   |   |   |   |   |   |   |   |   |
# +---+---+---+---+ +---+---+---+---+ +---+---+---+---+
# percept x,x,x,x,x
# score: -1

proc show agent { global map
    set lines ""; set divider "+---+---+---+---+"
    set status "$map($agent,next)>$map($agent,x)$map($agent,y)"
    if { $map($agent,dir) == "X" } {
        set status "$map($agent,q)$map($agent,moves)" }
    if { $map($agent,dir) == "Q" } { set status "Q$map($agent,moves)" }
```

```

foreach y "4 3 2 1" { lappend lines $divider
  set line "|"
  foreach x "1 2 3 4" {
    set xy "$x,$y"
    set a " "; if [info exists map($agent,$xy)] { set a "." }
    if { "$map($agent,x),$map($agent,y)" == $xy } { set a $map($agent,dir) }
    set w " "; if [info exists map(w,$xy)] { set w "W"
      if [info exists map($agent,killed)] { set w "w" } }
    set p " "; if [info exists map(p,$xy)] { set p "P" }
    set g " "; if [info exists map($agent,g,$xy)] { set g "G" }
    append line "$a$w$p$g|"
  }
  lappend lines $line
}
lappend lines $divider
set line "$agent $map($agent,score) $status $map($agent,pname)"
lappend lines [format %-.21.21s $line]
join $lines "\n" }

#####
proc v args { return 0 } ;# verbosity low
proc v args { return 1 } ;# verbosity high

#-----
# this assumes all block lines are the same length
proc showAll agents {
  uplevel { set outputAll ""; set showAllWct 0; set outputLine ""; global width
    foreach id $agents { set outputLine [merge $outputLine [show $id]]
      if { [incr showAllWct] % $width == 0 } {
        lappend outputAll $outputLine; set outputLine "" } }
    if { $outputLine != "" } { lappend outputAll $outputLine }
    join $outputAll "\n" } }
proc merge {block1 block2} {
  set output ""
  set max 0; foreach line1 [split $block1 "\n"] {
    set len [string length $line1]; if { $len > $max } { set max $len } }
  if { $max == 0 } { return $block2 }
  foreach line1 [split $block1 "\n"] line2 [split $block2 "\n"] {
    lappend output "[format %-${max}s $line1] [format %-.21.21s $line2]" }
  join $output "\n" }

```

```
#####
# subroutines
#####

proc average args {
    if { [llength $args] == 1 } { set args [lindex $args 0] }
    set sum 0; set count 0
    foreach arg $args { catch { set sum [expr $sum + $arg]; incr count } }
    if { $count == 0 } { return 0 }
    expr 1.0 * $sum / $count }

#-----
# modified from expect's mkpasswd by Don Libes
proc rand args {
    set fileId [open /dev/urandom r]
    binary scan [read $fileId 4] i1 number
    close $fileId
    return $number }

#-----
# pseudo-random number generator
proc randomSeed seed { global RNDseed version
    set RNDseed $seed; set version $seed }
proc random15 {} { global RNDseed; # 15 bit int: 0..32767
    set RNDseed [expr $RNDseed * 1103515245 + 12345]; # overflows at 32 bits
    expr int ( $RNDseed / 65536 ) % 32768 }
proc random {low high} {
    expr int ( $low + [random15] * ($high - $low + 1) / 32768) }

#-----
# pick one at random
proc pick args {
    if { [llength $args] == 1 } { set args [lindex $args 0] }
    lindex $args [random 0 [expr [llength $args] - 1]] }

#-----
# permute a list and return it
proc permute args { set out "";
    if { [llength $args] == 1 } { set args [lindex $args 0] }
```

```

while { [llength $args] > 0 } {
    set nexti [random 0 [expr [llength $args] - 1]];
    lappend out [lindex $args $nexti];
    set args [lreplace $args $nexti $nexti];
}; return $out;
}

#####
# main program
#####
proc main args { global agents map verbose keep sc
#####
# foreach agent $agents { putsv 0 "agent is $agent" }
# agent is either "<" ">" "^" or "V" to show directionality, or "x" if dead
foreach key [array names map] { if ![regexp "name$" $key] { unset map($key) } }
puts [array get map]
# catch { unset map }
foreach agent $agents {
    set map($agent,dir) ">"
    set map($agent,x) 1
    set map($agent,y) 1
    set map($agent,score) 0
    set map($agent,moves) 0
    set map($agent,perc) "x,x,x,x,x"
    set map($agent,next) "@" } ;# "start"
# only show debug lines if there is exactly one agent
if { [llength $agents] != 1 } { proc putsv1 args { } }

#####
set matype "r"; # stub
if { $matype == "r" } { global seed
    putsv 1 "Generating a Random Map, using seed $seed"
    set cells "1,3 1,4 2,3 2,4 3,1 3,2 3,3 3,4 4,1 4,2 4,3 4,4"
    foreach cell $cells {
        if { [random15] % 5 != 0 } continue
        set map(p,$cell) 1
        regexp {(.)} $cell foo x y
        set map(b,$x,$y) 1; # assign breeze
        set map(b,$x,[expr $y-1]) 1
        set map(b,$x,[expr $y+1]) 1
    }
}

```

```

    set map(b,[expr $x-1],$y) 1
    set map(b,[expr $x+1],$y) 1
  }
  set w [pick $cells]; set map(w,$w) 1; # hide the wumpus
  regexp {(.),(.)} $w foo x y
  set map(s,$x,$y) 1; # assign stench
  set map(s,$x,[expr $y-1]) 1
  set map(s,$x,[expr $y+1]) 1
  set map(s,[expr $x-1],$y) 1
  set map(s,[expr $x+1],$y) 1
  set g [pick $cells] ;# hide the gold
  foreach agent $agents { set map($agent,g,$g) 1 }
}

#####

# debugging information
putsv 2 [lsort [array names map]]

log_user 0; # stop spawned process output from appearing on screen

# start each agent
foreach agent $agents {
  putsv 1 "starting $agent $map($agent,name) ($map($agent,pname))"
  # if it was open, close it so we can reopen it
  if [info exists keep($agent.sid)] { set spawn_id $keep($agent.sid)
    # puts "killing $spawn_id $keep($agent.pid)"
    # exec kill $keep($agent.pid)
    catch { close }
    wait -nowait }
  # catch { set spawn_id $keep($agent.sid); close; wait -nowait }
  set keep($agent.pid) [eval spawn $map($agent,name)]
  set keep($agent.sid) $spawn_id
  # puts "spawning $spawn_id $keep($agent.pid)"

  # calculate the percept: stench,breeze,glitter,bump,scream
  set xy "$map($agent,x),$map($agent,y)"
  set percept ""
  if { [info exists map(s,$xy)] } {
    append percept "y," } else { append percept "n," }
}

```

```

    if { [info exists map(b,$xy)] } {
        append percept "y," } else { append percept "n," }
    if { [info exists map($agent,g,$xy)] } {
        append percept "y," } else { append percept "n," }
    append percept "n,n"; # guaranteed true on first move
    set map($agent,perc) $percept
}
puts [showAll $agents]

while 1 {
    # wait for dungeonmaster to press enter or type a number
    putsv 1 "press ENTER to continue (or enter a number)"
    if { $verbose > 0 } { set turns [gets stdin] } else { set turns 1 }
    if [regexp {[qx]} [string tolower $turns]] break

    # take the number of moves entered, default is 1, not less than 1.
    if ![regexp {[1-9][0-9]*$} $turns] { set turns 1 }
    while { $turns > 0 } {
        incr turns -1
        # give each agent a percept and get its move
        set alive 0
        foreach agent $agents {
            if { $map($agent,dir) == "X" } continue ;# agent died
            if { $map($agent,dir) == "Q" } continue ;# agent quit
            incr alive; # count remaining players

            set spawn_id $keep($agent.sid)

            set percept $map($agent,perc)
            set det "stench,breeze,glitter,bump,scream" ;# details
            putsv1 1 "To $agent: sending percept ($det)=$percept"
            send "$percept\r"

            set got "X"; # quit
            set timeout 1
            # watch for crashes.
            if [catch { expect {
                -re {[r\n]*([r\n]+)[r\n]+} { set got $expect_out(1,string)
                    set got [string toupper $got]
                    # debug lines can be printed "# ... \n"

```

```

    set got [string trim $got]
    set pat {(S|SHOOT|L|LEFT|R|RIGHT|A|FORWARD|Q|QUIT|G|GRAB)$}
    if ![regexp $pat $got] {
        putsv1 1 "Fr $agent: $got"
        exp_continue -continue_timer }
    set map($agent,next) $got
    putsv1 1 "Fr $agent: $got"
}
timeout { putsv 1 "## $agent ($map($agent,moves)): timeout"
    set map($agent,q) "T"; set map($agent,dir) "X" }
eof { putsv 1 "## $agent ($map($agent,moves)): stopped (eof)"
    set map($agent,q) "E"; set map($agent,dir) "X" }
} } ] { putsv 1 "## $agent ($map($agent,moves)): died"
    set map($agent,q) "D"; set map($agent,dir) "X" }

if { $got == "QUIT" } { set got "Q" }
if { $got == "LEFT" } { set got "L" }
if { $got == "RIGHT" } { set got "R" }
if { $got == "FORWARD" } { set got "A" }
if { $got == "SHOOT" } { set got "S" }
if { $got == "GRAB" } { set got "G" }
set map($agent,next) $got

set x $map($agent,x)
set y $map($agent,y)
set xy "$x,$y"

if ![info exists map($agent,$xy)] {
    incr map($agent,score) $sc(RV) ;# reward for each room visited
    set map($agent,$xy) 1 } ;# visited
set dirGot "$map($agent,dir)$got"

# we don't count quitting as an action that costs sc(Mv) points
if { $got == "Q" } {
    if { $xy == "1,1" } { incr map($agent,score) $sc(QE) }
    putsv 1 "## $agent ($map($agent,moves)): quitting"
    set map($agent,q) "Q"; set map($agent,dir) "Q"; continue }

incr map($agent,moves) ;# count the number of moves made
incr map($agent,score) $sc(Mv) ;# penalty for each action

```

```

set killed "n"
if { $got == "S" } {
    if [info exists map($agent,shot)] continue
    set map($agent,shot) 1
    incr map($agent,score) $sc(Sh) ;# penalty for shooting arrow
    set x $map($agent,x); set y $map($agent,y)
    if { $dirGot == "<S" } { while { $x >= 1 } {
        if { [info exists map(w,$x,$y)] } {
            set killed "y"; break }; incr x -1 } }
    if { $dirGot == ">S" } { while { $x <= 4 } {
        if { [info exists map(w,$x,$y)] } {
            set killed "y"; break }; incr x 1 } }
    if { $dirGot == "^S" } { while { $y <= 4 } {
        if { [info exists map(w,$x,$y)] } {
            set killed "y"; break }; incr y 1 } }
    if { $dirGot == "vS" } { while { $y >= 1 } {
        if { [info exists map(w,$x,$y)] } {
            set killed "y"; break }; incr y -1 } }
    if { $killed == "y" } { set map($agent,killed) 1
        incr map($agent,score) $sc(KW) } ;# reward for killing wumpus
    }

if [info exists map($agent,g,$xy)] {
    putsv1 1 "## $agent: gold is here. YESSS!" }
if { $got == "G" } { # did we get the gold?
    if [info exists map($agent,g,$xy)] {
        unset map($agent,g,$xy) ;# can collect only once
        incr map($agent,score) $sc(Go) } ;# reward for gold
    }

if { $dirGot == "^L" } { set map($agent,dir) "<"; set dirGot "" }
if { $dirGot == "<L" } { set map($agent,dir) "v"; set dirGot "" }
if { $dirGot == "vL" } { set map($agent,dir) ">"; set dirGot "" }
if { $dirGot == ">L" } { set map($agent,dir) "^"; set dirGot "" }

if { $dirGot == "^R" } { set map($agent,dir) ">"; set dirGot "" }
if { $dirGot == "<R" } { set map($agent,dir) "^"; set dirGot "" }
if { $dirGot == "vR" } { set map($agent,dir) "<"; set dirGot "" }
if { $dirGot == ">R" } { set map($agent,dir) "v"; set dirGot "" }

```

```

set bump "n"
if { $dirGot == "^A" } { if { $y == 4 } {
    set bump "y" } else { incr map($agent,y) 1 }; set dirGot "" }
if { $dirGot == "vA" } { if { $y == 1 } {
    set bump "y" } else { incr map($agent,y) -1 }; set dirGot "" }
if { $dirGot == "<A" } { if { $x == 1 } {
    set bump "y" } else { incr map($agent,x) -1 }; set dirGot "" }
if { $dirGot == ">A" } { if { $x == 4 } {
    set bump "y" } else { incr map($agent,x) 1 }; set dirGot "" }
if { $bump == "y" } { incr map($agent,score) $sc(Bu) } ;# bump penalty

# in case we moved just now
set x $map($agent,x)
set y $map($agent,y)
set xy "$x,$y"

# calculate the percept: stench,breeze,glitter,bump,scream
set xy "$map($agent,x),$map($agent,y)"
set percept ""
if { [info exists map(s,$xy)] } { set s "y" } else { set s "n" }
if { [info exists map(b,$xy)] } { set b "y" } else { set b "n" }
if { [info exists map($agent,g,$xy)] } { set g "y" } else { set g "n" }
set map($agent,perc) "$s,$b,$g,$bump,$skilled"

# did we run into a live wumpus?
if { [info exists map(w,$xy)] && ![info exists map($agent,killed)] } {
    putsv 1 "## $agent ($map($agent,moves)): wumpus is here. AAAAAH!" }
if { ![info exists map($agent,killed)] && [info exists map(w,$xy)] } {
    incr map($agent,score) $sc(Di) ;# penalty for death
    set map($agent,q) "W"; set map($agent,dir) "X" }

# did we fall into a pit?
if [info exists map(p,$xy)] {
    putsv 1 "## $agent ($map($agent,moves)): pit is here. AAAAAH!"
    incr map($agent,score) $sc(Di) ;# penalty for death
    set map($agent,q) "P"; set map($agent,dir) "X" }
}
}
puts [showAll $agents]

```

```
    if { $alive == 0 } { putsv 1 "All wumpus agents have terminated"; break }
}

set scoreline ""
foreach agent $agents { lappend scoreline "$map($agent,pname)=$map($agent,score)" }
puts [join $scoreline " "]
foreach agent $agents { lappend keep($agent.scores) $map($agent,score) }
}
foreach agent $agents { set keep($agent.scores) "" }
if { $seed == "" } { set seed [rand] }
randomSeed $seed; set seedWas $seed
while { $iter > 0 } { main; incr iter -1 }
foreach agent $agents { if { [llength $keep($agent.scores)] == 1 } continue
    puts "$agent average score [average $keep($agent.scores)] - $map($agent,pname)" }
puts "Seed was $seedWas"
putsv 0 "Wumpus Driver Terminating"
```

## Appendix D

# Numbers Driver Source Code

Following is a version of the driver program that will be used to evaluate student programs.

```
#!/usr/bin/perl -w
# p2n Testbed
# p2n input is "phonemes" on argv
# p2n output is one line to stdout

if ( @ARGV < 2 ) {
    print "** Phoneme to Number Tester **\n";
    print "usage: nDriver worker file(s)\n";
    print "  worker is program to be tested\n";
    print "  . input is .phn file on standard in\n";
    print "  . output is one recognized word per line\n";
    print "  file(s) is a list of test files\n";
    print "example: nDriver Don1 ~don/corpus/*\n";
    exit;
}

$worker = shift @ARGV; # the student's program

# foreach file mentioned on ARGV do the conversion
```

```

# important words
foreach $word qw(
  a and double eight eighteen eighteenth eighth eightieth
  eighty eleven eleventh fifteen fifteenth fifth fiftieth
  fifty first five fortieth forty four fourteen fourteenth
  fourth half hundred hundredth nine nineteen nineteenth
  ninetieth ninety ninth oh one second seven seventeen
  seventeenth seventh seventieth seventy six sixteen sixteenth
  sixth sixtieth sixty ten tenth third thirteen thirteenth
  thirtieth thirty thousand three triple twelfth twelve
  twentieth twenty two zero
) { $vocab{$word} = 1 }

# # maybe words
# foreach $word qw(
#   dash hyphen number
#   area code
#   north south east west
#   avenue road street
#   o'clock
# ) { $vocab{$word} = 1 }

sub normalize { # text
  my ( $txt ) = @_ ;
  $txt0 = $txt; $txt0 =~ s/\n *$//;
  $txt =~ s/\n/ /g;
  $txt = " $txt ";
  $txt =~ s/[.][a-z]+//g; # .bn .ls
  # replace <bs:[a-z ]*> with space (background speech)
  $txt =~ s/<bs:[a-z ]+>/ /g
  $txt =~ s/<[a-z]+>/ /g; # replace <[a-z]*> with space
  # delete any utterance that has < > [ ] *
  if ( $txt =~ /[~a-z' ]/ ) {
    # print " # txt is ( $txt0 )\n";
    return "" }
  $txt =~ s/ +/ /g;
  $out = " "; foreach $w (split / +/, $txt) {
    $out .= "$w " if ( defined ( $vocab{$w} ) ) }
  return $out; # format is: " word word word "
}

```

```

sub consider {
    ( $file ) = @_;
    $base = $file;
    $base =~ s/.phn$//;
    $base =~ s/.txt$//;
    $base =~ s/.wrд$//;
    $base =~ s/.wav$//;
    return if -e "$base.p2nX"; # marked as impossible
    return if ( $done{$base} ); $done{$base} = 1;
    # print "doing $base\n";

    # find out what the human transcriber did
    $txt0 = 'cat $base.txt';
    $txt1 = normalize ( $txt0 );
    if ( $txt1 eq "" ) {
        $phons = " ";
        foreach $line ( split /\n/, 'cat $base.phn' ) {
            $line =~ s/[\r\n]+//;
            next if ( $line =~ /MillisecondsPerFrame:/ );
            next if ( $line =~ /END OF HEADER/ );
            if ( $line !~ /^(д+) (д+) (.*)/ ) {
print "weird: ($line)\n"; next }
            $phons .= "$3 "; }
        # print " # phn: ($phons) SKIP\n";
        return }

    # let the test program do its translation
    $res0 = '$worker < $base.phn';
    # ignore output lines that start with # (debug lines)
    $res1 = " "; foreach $line ( split /\n/, $res0 ) {
        next if ( $line =~ /^#/ ); $res1 .= "$line " }
    $res1 = normalize ( $res1 );

    $count++; # count this one
    if ( $res1 eq $txt1 ) { $okay++;
        $save = 0; if ( $count ) { $save = 100 * $okay / $count }
        $score = sprintf "%дd/%-дд (.1f%%)", $okay, $count, $save;
        print "$score okay $base ($res1)\n";
        return }
}

```

```

print "err $base\n";
# provide the phonemes for convenient comparison
$phons = " ";
foreach $line ( split /\n/, 'cat $base.phn' ) {
    $line =~ s/[\r\n]+//;
    next if ( $line =~ /MillisecondsPerFrame:/ );
    next if ( $line =~ /END OF HEADER/ );
    if ( $line !~ /^(\\d+) (\\d+) (\\.*)/ ) {
        print "weird: ($line)\n"; next }
    $phons .= "$3 "; }
print " # phn: ($phons)\n";
print " # tru: ($txt1)\n";
print " # stu: ($res1)\n";
print "###\n$res0\n###\n";
return; # unless you want Keep/Move

print "Keep or Move [Km]: ";
chomp ( $ans = <STDIN> );
if ( $ans eq "m" ) {
    print "moving $base to numbers/err/\n";
    print 'mv $base.* numbers/err'
}
}

$okay = 0; $count = 0;
foreach $arg (@ARGV) {
    foreach $file (glob $arg) { consider $file } }

$save = 0; if ( $count ) { $save = 100 * $okay / $count }
printf "score: $okay/$count (%.1f%%)\n", $save;

```

# Appendix E

## Test Bank

### Test Bank

- 1: (p.2) What does AI stand for?
- 2: (p.8) Find  $p(A \cap \bar{B})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .
- 3: (p.8) Find  $p(\bar{A} \cap B)$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .
- 4: (p.9) Find  $p(\bar{A} \cap \bar{B})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .
- 5: (p.9) Find  $p(A|B)$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .
- 6: (p.9) Find  $p(A|\bar{B})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .
- 7: (p.9) Find  $p(B|A)$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .
- 8: (p.9) Find  $p(B|\bar{A})$  given  $p(A)=1/3$ ,  $p(B)=7/18$ ,  $p(A \cap B)=1/9$ .
- 9: (p.9) Find  $p(\bar{A} \cap B)$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .
- 10: (p.9) Find  $p(\bar{A} \cap \bar{B})$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .
- 11: (p.9) Find  $p(A|B)$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .
- 12: (p.10) Find  $p(A|\bar{B})$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .
- 13: (p.10) Find  $p(B|A)$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A \cap B)=7/12$ .

- 14:** (p.10) Find  $p(B|\bar{A})$  given  $p(A)=5/6$ ,  $p(B)=2/3$ ,  $p(A\cap B)=7/12$ .
- 15:** (p.10) Find  $p(A\cap\bar{B})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .
- 16:** (p.10) Find  $p(\bar{A}\cap B)$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .
- 17:** (p.10) Find  $p(\bar{A}\cap\bar{B})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .
- 18:** (p.10) Find  $p(A|B)$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .
- 19:** (p.10) Find  $p(A|\bar{B})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .
- 20:** (p.11) Find  $p(B|A)$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .
- 21:** (p.11) Find  $p(B|\bar{A})$  given  $p(A)=3/7$ ,  $p(B)=3/7$ ,  $p(A\cap B)=2/21$ .
- 22:** (p.13) What is the product rule?
- 23:** (p.13) What is Bayes' rule?
- 24:** (p.19) Resolve:  $(-a\ c)\ (-b\ a)\ (-c\ a)\ (-c\ b)\ (b\ c)$
- 25:** (p.20) Resolve:  $(-a\ b)\ (-b\ c)\ (a\ b)$
- 26:** (p.20) Resolve:  $(-c\ -d\ a)\ (a\ b\ d)\ (a\ c)\ (a\ c\ d)$
- 27:** (p.20) Resolve:  $(-b\ -c\ a)\ (-c\ b\ d)\ (-d\ a\ b)$
- 28:** (p.20) Resolve:  $(-a\ d)\ (-b\ c)\ (-c\ a)\ (-d\ a\ c)\ (a\ b)$
- 29:** (p.20) Resolve:  $(-b\ a)\ (-b\ a\ c)\ (-c\ -d\ a)\ (a\ b\ c)$
- 30:** (p.20) Resolve:  $(-a\ -b\ c)\ (-b\ -c\ d)\ (-b\ a)$
- 31:** (p.20) Resolve:  $(-a\ -c\ b)\ (-a\ -d\ c)\ (-b\ d)\ (-c\ a\ b)$
- 32:** (p.20) Resolve:  $(-c\ -d\ a)\ (-d\ b\ c)\ (a\ b)\ (a\ c\ d)\ (c\ d)$
- 33:** (p.21) Resolve:  $(-a\ -d\ c)\ (-d\ a)\ (a\ b)$
- 34:** (p.39) Discuss: 1.0 Artificial Intelligence
- 35:** (p.39) Discuss: 1.1 Turing test
- 36:** (p.39) Discuss: 1.1 automated reasoning
- 37:** (p.39) Discuss: 1.1 machine learning

- 38:** (p.39) Discuss: 1.1 total Turing test
- 39:** (p.39) Discuss: 1.1 agent
- 40:** (p.40) Discuss: 1.2 dualism
- 41:** (p.40) Discuss: 1.2 materialism
- 42:** (p.40) Discuss: 1.2 logical positivism
- 43:** (p.40) Discuss: 1.2 algorithm
- 44:** (p.40) Discuss: 1.2 Gödel's incompleteness theorem
- 45:** (p.40) Discuss: 1.2 intractability
- 46:** (p.40) Discuss: 1.2 NP-completeness
- 47:** (p.40) Discuss: 1.3 machine evolution
- 48:** (p.40) Discuss: 1.3 genetic algorithms
- 49:** (p.40) Discuss: 1.3 expert systems
- 50:** (p.40) Discuss: 1.3 frames
- 51:** (p.40) Discuss: 2.2 rational agent
- 52:** (p.40) Discuss: 2.2 autonomous agent
- 53:** (p.40) Discuss: 2.3 simple reflex agent
- 54:** (p.40) Discuss: 2.3 goal-based agent
- 55:** (p.40) Discuss: 2.3 utility-based agent
- 56:** (p.41) Discuss: 2.4 accessible environment
- 57:** (p.41) Discuss: 2.4 deterministic environment
- 58:** (p.41) Discuss: 2.4 episodic environment
- 59:** (p.41) Discuss: 2.4 static vs dynamic environment
- 60:** (p.41) Discuss: 2.4 discrete vs continuous environment
- 61:** (p.41) Discuss: 3.x search

- 62:** (p.41) Discuss: 3.x path cost
- 63:** (p.41) Discuss: 3.x breadth-first search
- 64:** (p.41) Discuss: 3.x uniform-cost search
- 65:** (p.41) Discuss: 3.x depth-first search
- 66:** (p.41) Discuss: 3.x depth-limited search
- 67:** (p.41) Discuss: 3.x iterated deepening search
- 68:** (p.41) Discuss: 3.x bidirectional search
- 69:** (p.41) Discuss: 4.x heuristics
- 70:** (p.41) Discuss: 4.x best-first search
- 71:** (p.41) Discuss: 4.x greedy search
- 72:** (p.42) Discuss: 4.x A\* search
- 73:** (p.42) Discuss: 4.x iterated improvement
- 74:** (p.42) Discuss: 6.x knowledge representation
- 75:** (p.42) Discuss: 6.x inference (sound, complete)
- 76:** (p.42) Discuss: 6.x propositional logic
- 77:** (p.42) Discuss: 7.x first-order logic
- 78:** (p.42) Discuss: 7.x atomic sentence
- 79:** (p.42) Discuss: 7.x predicate
- 80:** (p.42) Discuss: 7.x quantified sentence
- 81:** (p.42) Discuss: 7.x situational calculus
- 82:** (p.42) Discuss: 7.x diagnostic rules
- 83:** (p.42) Discuss: 7.x causal rules
- 84:** (p.42) Discuss: 9.x unification
- 85:** (p.42) Discuss: 9.x Modus Ponens

- 86: (p.42) Discuss: 9.x Horn form
- 87: (p.42) Discuss: 9.x resolution
- 88: (p.43) Discuss: 9.x conjunctive normal form
- 89: (p.43) Discuss: 9.x implicative normal form
- 90: (p.43) Discuss: 13.x conditional plans
- 91: (p.43) Discuss: 13.x execution monitoring
- 92: (p.43) Discuss: 13.x action monitoring
- 93: (p.43) Discuss: 13.x replanning agent
- 94: (p.43) Discuss: 14.x prior probabilities
- 95: (p.43) Discuss: 14.x conditional probabilities
- 96: (p.43) Discuss: 14.x joint probability distribution
- 97: (p.43) Discuss: 14.x Bayes' rule
- 98: (p.43) Discuss: 14.x conditional independence
- 99: (p.43) Discuss: 14.x Bayesian updating
- 100: (p.43) Discuss: 15.x belief networks
- 101: (p.43) Discuss: 15.x stochastic simulation
- 102: (p.43) Discuss: 15.x truth-functional system
- 103: (p.43) Discuss: 18.x performance element
- 104: (p.44) Discuss: 18.x learning element
- 105: (p.44) Discuss: 18.x inductive learning
- 106: (p.44) Discuss: 19.x neural network
- 107: (p.44) Discuss: 19.x perceptron
- 108: (p.44) Discuss: 19.x linearly separable function
- 109: (p.44) Discuss: 19.x feed-forward network

- 110: (p.44) Discuss: 19.x back-propagation
- 111: (p.44) Discuss: 19.x Bayesian learning
- 112: (p.44) Discuss: 19.x multi-layer feed-forward network
- 113: (p.44) Discuss: 22.x speech act
- 114: (p.44) Discuss: 22.x phrase-structure grammar
- 115: (p.44) Discuss: 22.x context-free grammar
- 116: (p.44) Discuss: 22.x encoded message
- 117: (p.44) Discuss: 22.x situated language
- 118: (p.44) Discuss: 22.x augmented grammar
- 119: (p.44) Discuss: 22.x pragmatic interpretation
- 120: (p.45) Discuss: 22.x disambiguation
- 121: (p.45) Discuss: 24.x image formation
- 122: (p.45) Discuss: 24.x image processing
- 123: (p.45) Discuss: 27.1 anytime algorithm
- 124: (p.45) Discuss: 27.2 bounded optimality
- 125: (p.45) Discuss: 27.2 prisoner's dilemma

# Index

- a priori probabilities, 13
- ACM, 46
- bake off, 24
- basis, 35
- Bayes' rule, 12
- CNF, 17, 19
- complementary literals, 18
- conjunction, 7, 17, 19
- conjunctive normal form, 19
- CSLU, 33
- disjunction, 7, 17, 19
- first order logic, 16
- IEEE-CS, 46
- induction, 35
- modus ponens, 16
- numbers, 32
- over training, 36
- percept, 25
- phonemes, 32
- Predicate Calculus, 16
- prior probabilities, 13
- probability, 6
- product rule, 12
- Propositional Logic, 16
- resolution, 17
- Robinson, John Alan, 17
- rules of inference, 16
- syllogisms, 16
- tautology, 18
- vacuum, 25
- Venn diagram, 7, 8
- Wumpus, 29