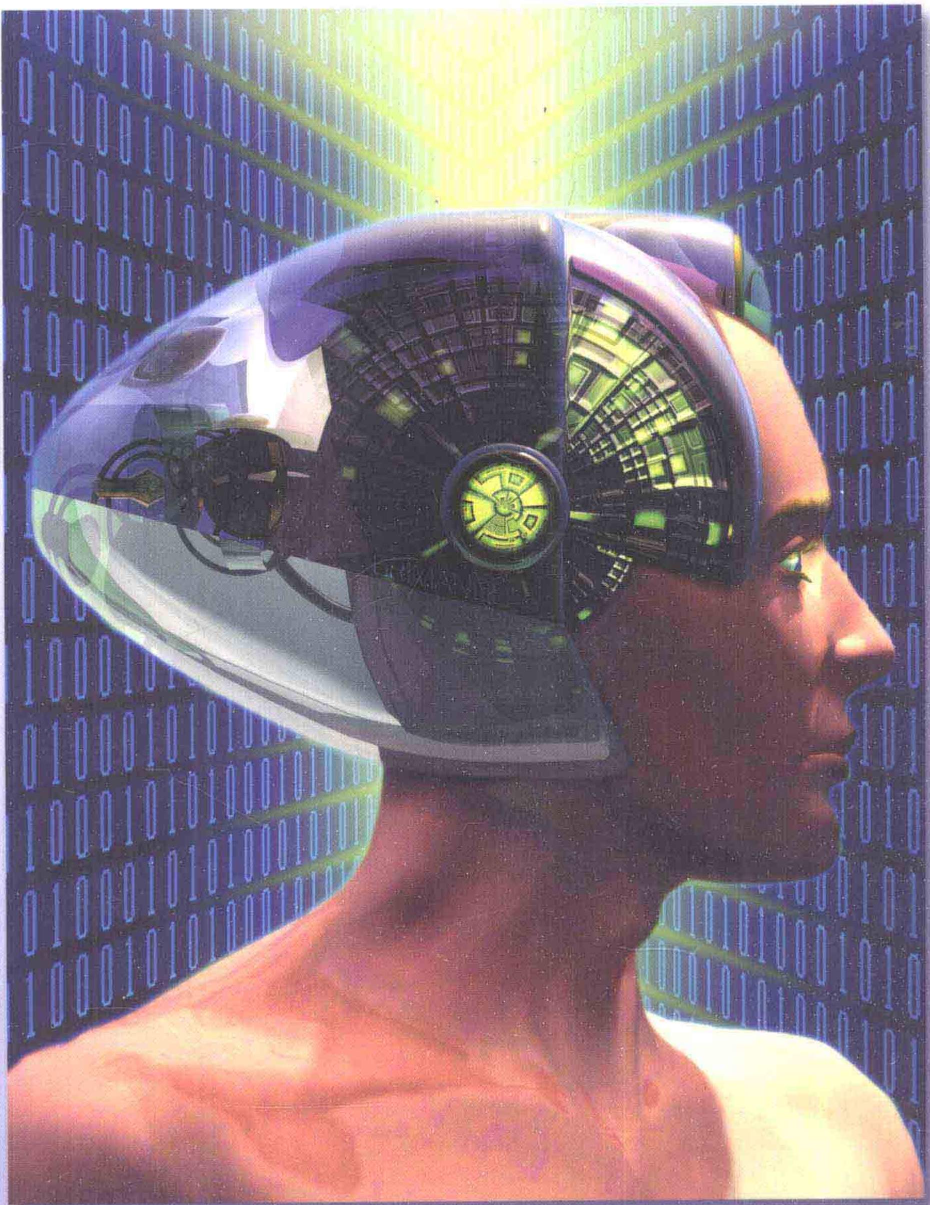
Hands-On AI with Java

Smart Gaming, Robotics, and More

Build programs that work intelligently with humans

Add smarts to computer games

Program optimal path-finding for machining and robotics



Edwin Wise

TAB ELECTRONICS

Hands-On AI with Java
Smart Gaming, Robotics, and More

Edwin Wise

McGraw-Hill
New York Chicago San Francisco Lisbon
London Madrid Mexico City Milan New Delhi
San Juan Seoul Singapore Sydney Toronto

2 3 4 5 6 7 8 9 0 DOC/DOC 0 1 9 8 7 6 5 4

ISBN 0-07-142496-2

The sponsoring editor for this book was Judy Bass and the production supervisor was Pamela A. Pelton. It was set in Century Schoolbook by the Alden Group, Oxford. The art director for the cover was Anthony Landi.

Printed and bound by RR Donnelley.

This book is printed on recycled, acid-free paper containing a minimum of 50% recycled, de-inked fiber.

ABOUT THE AUTHOR

Edwin Wise is Vice President of Development and partner at Wittlock Engineering, LLC, Austin, Texas. A software engineer with over 20 years of experience, his expertise and interests range from basic electronics and software development to microcontrollers, AI, and robots. He is the author of *Applied Robotics; Applied Robotics II*; and *Animatronics: A Guide to Animated Holiday Displays*, all published by Delmar Learning. He lives in Austin.

ACKNOWLEDGMENTS

This book could not have been written without the support of my wife, Marla. And, of course, McGraw-Hill would not have asked me to write it if I didn't have the support of you, my readers.

CONTENTS

CHAPTER 1

Artificial Intelligence

This chapter introduces the form and format of this book and the topic of Artificial Intelligence. The bulk of this chapter sets some groundwork in the form of the display classes *JGraphPanel, JLineChart,* and *JHintonGraph*, and introduces the complex topic of validating the results of your AI projects.

Introduction

What is artificial intelligence (AI)? Even limiting our scope to the world of computer science, this is a question that has different answers to different people.

You can pick up one book on AI techniques and it will contain a hundred and one ways to perform supervised learning in backpropagation networks. A different book might list the tasks and technologies required to build expert systems using predicate logic. Other books will focus on categorization of input data using self-organizing networks. Yet another programmer may eschew all of these and use state machines to manage their problem.

On a higher level, AI topics include vision processing, speech recognition, path planning, expert problem solving systems, robot guidance systems, etc. There are few areas of computer science that are untouched by AI if you paint with a broad enough brush.

In general, AI techniques are used when a direct numerical solution is not available or feasible. AI gets to attack the "hard" problems.

The philosophical landscape of AI is littered with people debating whether machines can ever be truly intelligent. That is an interesting question, and the cognitive scientists and computer science researchers who are trying to simulate or emulate actual brains have their work cut out for them. However, if you approach AI from a practical level things become much simpler. We do not care if it is truly intelligent, only that it solves our problem. Most application developers simply want an algorithm that will solve the problem in hand, and AI techniques are good at solving hard problems.

While most books will pick one or two types of AI, this book attempts to look at most of the techniques within the AI umbrella. This means our focus is broad rather than deep. This also means that we will only be able to cover the *essence* of most techniques and not delve into all of the details. On the other hand, once you have the basics in place it becomes much easier to read and apply the dizzying variety of details available in the research papers and in other books.

The content of this book embodies my philosophy of books, the three book theory (TBT). The TBT states that, for any given technology or problem, you should have three books that address it. One book can be mistaken, confusing, or leave out important details. Three books, however, can give you a clear picture. So this book is not designed to be the *one true book*, but simply one of the three books on AI you need to get the job done. This is the overview and introduction to the building blocks of AI.

The first two chapters of this book lay some groundwork.

Chapter 3 begins exploring the outer edges of AI with we investigate reflexive control systems such as PID controls and fuzzy logic.

State-based systems provide scripted responses to stimulus. These include finite-state machines and their variations, Markov chains, and even Chatbots, explored in Chapter 4.

Many applications of AI do not look "intelligent" at all, but these problems can still be quite difficult to solve. A common problem, for example, is searching for something in a large data structure, decision tree, or problem space. The obvious methods of searching, such as visiting every possible answer to see if it is the best, can be too slow. A classic example is searching for the best next move in a chess game; there are simply too many possibilities to consider. Even though it would be *possible* to visit every single one it would not be practical. These problems are where smart search algorithms come into play, as detailed in Chapter 5.

Chapter 6 takes searching further, looking at search problems where it is not even possible to visit every possible solution. More intricate search algorithms are developed here, such as reinforcement learning and genetic algorithms.

In additional to searching techniques, "classical" AI is interested in the logical approach to thinking and problem solving, and the clever manipulation of symbols. We take a look at some of these logical techniques in Chapter 7.

Sub-symbolic techniques, however, have been the darling of the AI community for a while now. They provide an entirely different way of solving difficult categorization and prediction problems. Chapter 8 looks at neural networks and supervised neural network learning methods, while Chapter 9 expands on this base with unsupervised learning methods, the self-organizing maps.

Audience

This book is written primarily for the busy programmer though it can be used by anyone with an interest in learning how different AI techniques work.

It assumes the reader has a working knowledge of Java and is already proficient at programming. The details in this book dwell on the AI aspects of the problem and not on the way it was coded.

You should also be capable of working algebra problems, and not be frightened of mathematical symbols, such as $\sum$.

Purpose

This book is a field-guide to the world of artificial intelligence programming. It describes a broad range of AI tools that can each be applied to a wide array of problems.

These techniques range from those that can barely be considered intelligent, such as efficient tree searching, to classical AI symbolic techniques, to the bio-mimetic neural networks.

There are many ways to write about this kind of information, with each approach fulfilling a different need. The form of this book reflects my own deep-seated love of seeing how things work. It takes a practical approach to the subject, answering the question "how does this work?" rather than "why does this work?"

Examples

Most of this book describes the algorithms of AI, but there are also quite a few words devoted to describing the code examples. When I was learning programming I found the best teacher was existing code, to see how the principles are applied in practice. Working code can clarify a problem in a way that prose cannot. At the same time, it is important to know the theory behind the code for it to make sense. I hope I found the correct balance between theory and practice.

As the title of the book suggests, all of the code in this book is written in Java. Java is used because it is clear, simple, and portable. The downside that it is a moderately slow language is minor—once an algorithm is understood it can be adapted to the language of your choice.

The Java style used in the examples does not match the Sun Microsystems, Inc. coding conventions (as described at `java.sun.com/docs/codeconv`). I write in a mix of styles that have accreted over many years of programming.

Likewise, the documentation in the source is almost but *not quite* in JavaDoc format. I have found that Doxygen (`www.doxygen.org`) uses a more efficient commenting style and it provides marvelous documentation results. Plus, it can be applied to almost any language. I can just hear the comments now, however. "Why use a standard language like Java but then ignore the standard JavaDoc?" An excellent question, I am glad you asked. Java makes for great example code and can be run on any platform regardless of the documentation style. Doxygen is, I feel, a better format than JavaDoc—it is more readable and it does more than JavaDoc does. I am simply trying to use the best tools that I can.

All of the source and documentation can be found on the accompanying CD, as well as the website www.Simreal.com. Relevant excerpts are printed in the book, though to include all of the code in print would make the text cumbersome—there are many details of a working program that do not make for interesting reading.

Once you have the source code on your computer you can re-format it to match your own tastes using one of the many Java formatting tools, such as Jalopy (`jalopy.sourceforge.net`).

All of the code in this book was written for, and runs on, Java 1.4.1 using the JCreator Pro IDE (`www.jcreator.com`) under Microsoft Windows. Though it is not visible in the finished product, debugging support was done through the ubiquitous *System.out.println()* and, when the going got tough, JSwat (`www.bluemarsh.com/java/jswat/`).

If you want to explore the shape of the different equations in this text, I recommend the graphing tool *Equation Grapher*, found at `www.mfsoft.com/equationgrapher`.

Categories of AI

There are many different ways to divide and categorize AI systems. Some of these facets of AI are explored here.

You can separate the artificial intelligence field by application: computer vision, speech recognition, theorem proving, text parsing, adaptive controllers, and so forth. It may be more useful for a reference work to divide

AI up by technique: filters, controllers, tree search, state space search, state machines, pattern categorization, prediction in time-series data, etc.

Different AI techniques apply in different contexts. Some systems are fine when they can work offline, spending significant amounts of CPU time crunching on a problem. For real-time, online problems, different techniques may need to be used.

When an AI module has to learn or adapt to its environment, is that learning done offline, supervised by the programmer who feeds it inputs and answers, or is it online, learning as it goes? Other methods are static; they are programmed to behave in a particular manner and that is what they do, every time, all the time.

In my manufacturing applications I like the results to be repeatable and predictable, so I use deterministic methods whenever possible. When the customer runs a job and gets a weird result, I can repeat that run exactly and get the same result. But some AI techniques do not work that way; randomness enters into their calculations and the end results, while consistent across runs, may not be exactly the same. The benefit of these random (stochastic) algorithms is that they can find good solutions for very large or very difficult problems.

Whether you get exact results or not can also depend on the state space of the problem. Can each possible answer be listed and then searched (in theory if not in practice), making the problem space discrete? Or does the answer lie along a continuum, with an infinite possible number of answers, some more right than others? Sometimes you can adapt your problem to be either discrete or continuous. Sometimes you have to play the hand you are dealt.

Finally, the biggest rift in the AI system is the debate between symbolic and sub-symbolic systems. Is the problem defined using logic and inferences, symbols and manipulations of those symbols? Or is the problem solved using a network of interacting, low-level computational elements, with no explicit knowledge? The sub-symbolic (also known as connectionist) solutions are typically based on biological systems.

As you read through the various algorithms in this book, notice which categories they fit:

- *Task:* Filtering, control, searching, classification, prediction, or optimization?
- *Style:* Symbolic, connectionist, or collaborative?
- *Problem type:* Discrete or continuous?
- *Solution type:* Deterministic or stochastic?

- *Learning:* Static or adaptive? If adaptive, online or offline learning?
- *Execution:* Fast or slow? Offline or online?

This list is not comprehensive. For example, it is missing the architecture dimension; is the solution a sense–plan–act cycle, or is it a set of experts working in parallel, reacting to their environment in a behavioral system? The list goes on.

Validation

For most deterministic problems, validating your solution is a simple enough task. You present the program with a set of data and check the results against known answers. For example, searching a data structure is known to work when it finds an answer when an answer exists, and it does not find an answer when one does not exist.

But even then, you may want to quantify the results of the search in terms of how *quickly* the answer was found.

Once you move into the realms of stochastic solutions (those that involve randomness), you will get degrees of "rightness" or some percentage of right answers. In these cases, you need to apply statistical tools to get an idea of whether your solution is *right enough* for production work, or if you need to develop it further.

Finally, when you are adjusting and improving your AI code, it helps to know whether the change you just made has improved or degraded your algorithm, or if the change was not significant either way.

While this is not a book on statistics, and cannot go into much depth on that subject, we touch on some statistical techniques you can use as you explore different AI solutions.

In the process of exploring various validation techniques, we lay down some basic Java code for displaying information. Visual feedback is an excellent way to get a gut feeling for how something is behaving, and these classes come in handy later as we actually do work.

Statistics

There are two kinds of statistical techniques, descriptive and inferential.

Descriptive statistics provide numbers that can be used to characterize a set of data. In this case, we have all of the information in hand