Preface

Changes for the third edition

2. Replaces burning building example in Chapter 5 with lighting effects example.
3. Replaces fly-collision example in Chapter 6 with guessing-game example.
4. Adds new exercises and projects in Chapters 1, 3, 5, 6, 7, 9, and 10, particularly with more games.
5. Adds new Tips & Techniques on using groups in an Object Tree (see Chapter 9).
6. Updates Alice 2.2 with example worlds on DVD. Prevents name collisions of objects in a group.
7. Provides updates to Appendix B regarding exporting files and videos.
8. Adds Section 11.2, introducing Alice 3 beta.

Changes for the second edition

2. Exchanges and expands the content of Tips & Techniques for Chapter 3 and the content of Tips & Techniques for Chapter 8.
3. Replaces the content of Tips & Techniques in Chapter 5 with an expanded version of the “Creating Your Own People Models” subsection from Appendix B.
4. Moves the Summary of each chapter to appear immediately before the Exercises and Projects subsection.
5. Adds terms and concept questions to the Exercises and Projects subsections of each chapter.
6. Includes a brief description of flowchart symbols in Chapter 1 and simple flowcharts with selected examples in Chapters 1–4.
7. Adds a subsection in Appendix B on exporting an animation program to a file in QuickTime™ format.

“... what is the use of a book,” thought Alice, “without pictures or conversation?”

This book and the associated Alice system take an innovative approach to introductory programming. There have been relatively few innovations in the teaching of programming in the last 30 years, even though such courses are often extremely frustrating to students. The goal of our innovative approach is to allow traditional programming concepts to be more easily taught and more readily understood. The Alice system is free and is available at www.alice.org.

What should a programming course teach?

While many people have strong opinions on this topic, we feel there is a strong consensus that a student in a programming course should learn the following:

• Algorithmic thinking and expression: being able to read and write in a formal language.
• Abstraction: learning how to communicate complex ideas simply and to decompose problems logically.

• Appreciation of elegance: realizing that although there are many ways to solve a problem, some are inherently better than others.

What is different about our approach?

Our approach allows students to author on-screen movies and games, in which the concept of an “object” is made tangible and visible. In Alice, on-screen objects populate a 3D micro world. Students create programs by dragging and dropping program elements (if/then statements, loops, variables, etc.) in a mouse-based editor that prohibits syntax errors. The Alice system provides a powerful, modern programming environment that supports methods, functions, variables, parameters, recursion, arrays, and events. We use this strong visual environment to support either an objects-first or an objects-early approach (described in the ACM and IEEE-CS Computing Curricula 2001 and the Programming Fundamentals Core of the Computer Science Curriculum 2008 Interim Report) with an early introduction to events. In Alice, every object is an object that students can visibly see! We introduce objects in the very first chapter.

In our opinion, four primary obstacles to introductory programming must be overcome:

1. **The fragile mechanics of program creation, particularly syntax** The Alice editing environment removes the frustration of syntax errors in program creation and allows students to develop an intuition for syntax, because every time a program element is dragged into the editor, all valid “drop targets” are highlighted.

2. **The inability to see the results of computation as the program runs** Although textual debuggers and variable watchers are better than nothing, the Alice approach makes the state of the program inherently visible. In a sense, we offload the mental effort from the student’s cognitive system to his or her perceptual system. It is much easier for a student to see that an object has moved backward instead of forward than to notice that the “sum” variable has been decremented, rather than incremented. Alice allows students to see how their animated programs run, affording an easy relationship of the program construct to the animation action. Today’s students are immersed in a world where interactive, three-dimensional graphics are commonplace; we try to leverage that fact without pandering to them.

3. **The lack of motivation for programming** Many students take introductory programming courses only because they are required to do so. Nothing will ever be more motivating than a stellar teacher, but the right environment can go a long way. In pilot studies of classes using Alice, students do more optional exercises and are more likely to take a second class in programming than control groups of students using traditional tools. The most common request we received regarding earlier versions of Alice was to be able to share creations with peers; we have added the ability to run Alice programs in a World Wide Web browser so students can post them on their Web pages. Although we have seen increased motivation for all students, we have seen especially encouraging results with underrepresented groups, especially female students.

4. **The difficulty of understanding compound logic and learning design techniques** The Alice environment physically encourages the creation of small methods and functions. More importantly, the analogy of making a movie allows us to utilize the concept of a storyboard, which students recognize as an established movie-making process. We illustrate design techniques using simple sketches and screen captures. Also, we encourage the use of textual storyboards, progressively refining them and essentially designing with pseudocode.
How to use this text

Of course, as an instructor, you should use this text as you see fit! We list four ways we imagine the book being used, but you may discover others:

As the only text in a short or semester-long course on programming. This would allow students to build relatively complex (say, 300-line) programs by the end of the semester or term. Such a course might be for non-majors who want to learn the concepts behind programming without needing to transition to a real-world language. Alternatively, this course can be used as a pre-CS1 course for students who might like to major in computing but lack previous programming experience. In our NSF-supported study (NSF-0126833), we found that students who jump right into a rigorous CS1 course with little or no previous programming have an extremely high attrition rate. The use of Alice in a pre-CS1 course has significantly reduced attrition for these students in our CS1 courses.

As the first portion of a traditional “Introduction to Programming” course, such as CS1. Both Seymour Papert’s Logo and Rich Pattis’s Karel the Robot have been used this way, and these systems have inspired us greatly. Unlike these systems, Alice is powerful enough to support students for several semesters (for example, seniors majoring in computer science at Carnegie Mellon routinely write 3,000-line programs in Alice). However, many introductory programming courses must both teach concepts and also prepare students to write programs in traditional languages, such as Java. By learning Alice first, students become acquainted with the fundamental concepts of programming, and can quickly learn the specific syntax rules of a particular “real” language as a transition. The Alice environment can ease the transition by displaying programs with a Java-like syntax, as shown in Figure P-1-1.

As the programming component of a “Computer Literacy” course. At many schools, computer literacy courses attempt to give non-majors a broad introduction to computers and/or “information technology.” Many of these courses have removed their programming component and are little more than extended laboratories on “office productivity tools” such as spreadsheets and word processors. Alice has the potential to return a gentle programming component to computer literacy courses.

In a high school “Introduction to Programming” course. A course in Alice has great potential for a high school environment, where a high-interest, highly motivating environment is a teacher’s best friend. This book could be used as part of a stand-alone course or as preparation for the College AP computing course.

Instructional materials

An errata list, lecture notes, links to other instructional materials, and examples of course schedules can be found at www.aliceprogramming.net. Contact your local Pearson sales representative to get access to the instructor resources.

Structure of the book

Chapter 1 gives students some motivating reasons to want to write a computer program and addresses any fears they may have about programming (especially helpful for computer literacy courses). It then introduces some basic Alice concepts. A traditional, paper-based tutorial is presented in Appendix A. Computing and programming terminology (e.g., program, class, and object) are written in blue when first introduced and defined.

The remaining chapters each begin with a motivational overview of the chapter’s topic and end with exercises, projects, and a summary. The list below is a (very) brief overview of the major concepts covered, chapter by chapter. Clearly, the major focus of the text material is to introduce the fundamental concepts of programming.
Chapter 1  Getting started
Chapter 2  Design, stepwise refinement, and a first program
Chapter 3  Built-in functions, expressions, and simple control statements
Chapter 4  Object-oriented programming: methods, parameters, and inheritance
Chapter 5  Interactive programs: events and event handling
Chapter 6  Functions and conditional execution (If/Else)
Chapter 7  Definite (Loop) and indefinite loops (While)
Chapter 8  Repetition: recursion
Chapter 9  Lists
Chapter 10  Variables, revisiting inheritance, and array visualization
Chapter 11  Summary of fundamental concepts learned with Alice

We recommend a rapid pace through the first three or four chapters of the book (Chapters 1 and 2 can easily be covered in just 2 or 3 class days). Instructors should assign exercises selectively from the large number of exercises at the end of each chapter. A large number of ex-
Exercises were provided to allow the instructor to choose exercises most appropriate for their students. Examples of larger, more free-form “projects” are provided at the end of later chapters. (A “project” is a more advanced exercise that takes more time.)

Projects are meant to turn on the creative spirit, not weigh the student down. Where feasible, we recommend “open-ended” projects. An open-ended project is one which asks students to design their own animation beginning with their own storyline and using objects of their choosing. We do require that a project meet certain requirements—for example, “an interactive world, containing two or more interactions with the mouse, at least three methods, using a decision statement, and having objects from two classes you have created by writing class-level methods and saving out the new classes.” Alice lends itself particularly well to student demonstrations of their worlds to the rest of the class on the project due date.

Each chapter has a “Tips & Techniques” section. Collectively, these sections and Appendices A and B comprise a mini User’s Guide to Alice. The Tips & Techniques cover animation in Alice rather than traditional fundamental concepts of programming presented in the major chapter material. The techniques explained in these sections are strategically placed throughout the text, laying the groundwork for using these techniques in programming examples that follow. Tips & Techniques provide a guide for those who want to learn more about animation with Alice. Appendix A is a “getting started” tutorial. Appendix B describes how to manage the interface. Use of the interface is also integrated with text examples for programming concepts, where needed. The Tips & Techniques sections enrich the flavor of the book with selected “how-to” topics.

### Topic selection and sequence of coverage

The topic selection and sequence of coverage is in the instructor’s hands! The dependency chart in Figure P-1-2 should help you in selecting a path through the book.

From an overly simplistic perspective, Chapters 1–4 present topics in sequence and Chapters 5–11 are more independent and can be covered in many different sequences. A sequential coverage of Chapters 1–4 is an “Objects Early” approach. Skipping Chapter 3 is an “Objects First” approach. Examples in Chapters 4 and 5 were designed to be independent of Chapter 3. Thus, an Objects First approach can safely skip Chapter 3 and pick up these topics.
later as part of coverage of topics in Chapters 6 and 7. Another flexible sequencing option is to reverse Chapters 7 and 8 to allow coverage of recursion before Loop and While control structures.

In working with instructors in various high school, college and university settings, we found that topic selection and sequences are often based on time constraints for a particular course structure, pedagogy, and philosophy of teaching. If you are limited to three or four weeks as part of a larger course, you may wish to assign exercises throughout with only one project at the end. Also, you can use Tips & Techniques sections as reading assignments, not requiring classroom presentation time. The following Sequence Chart illustrates some examples of sequences.

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Strategy/time constraints</th>
<th>Courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>1–2–3–4 (and possibly 5)</td>
<td>Objects Early (3–4 weeks)</td>
<td>Computer Literacy First 3 weeks of CS1</td>
</tr>
<tr>
<td>1–2–4–5–6–7–8</td>
<td>Objects First (short course)</td>
<td>Introduction to Programming Pre-CS1</td>
</tr>
<tr>
<td>1–2–4–5–6–8–7</td>
<td>Objects First, Recursion Early (short course)</td>
<td>Pre-CS1</td>
</tr>
<tr>
<td>1–2–3–4–5 and selected topics from 6–11</td>
<td>Objects Early (full semester)</td>
<td>Introduction to Programming Pre-CS1</td>
</tr>
<tr>
<td>1–2–4–5–6 and selected topics from 7–11</td>
<td>Objects First (full semester)</td>
<td>Introduction to Programming Pre-CS1</td>
</tr>
</tbody>
</table>

Notes concerning specific aspects of the text

If you are using this text to teach/learn Alice without discussing design, you may skip the first section of Chapter 2. However, textual storyboards and stepwise refinement will be used throughout the rest of the text to provide a framework in which to discuss design from an algorithmic, problem-solving perspective. In response to dozens of requests from logic programming instructors, the second edition includes a brief explanation of flowchart symbols in Chapter 1 and several flowcharts inserted in Chapters 1–5. The flowcharts are purposely simple and are used to explain the underlying logic of selected example animation programs. We rely on the instructor to build on this basic approach for more complex examples. You may choose to use a different design framework (perhaps the Unified Modeling Language or a more traditional version of pseudocode). This may be done safely, without impacting the content.

In Chapter 4, we note that Alice does not provide a complete implementation of inheritance. When a new class is created in Alice, it gets a copy of the properties and methods of the base class and is saved in a new 3D model file. Subsequent changes to the super class are not reflected in the subclass.

Inheritance is accomplished in object-oriented programming languages via two mechanisms: (a) adding methods (behavior), and (b) adding extra state information via the use of mutable variables. We separate discussion of these mechanisms, introducing added behavior in Chapter 4, but deferring discussion of additional state information until Chapter 10. Because mutable variables are not visible/visual in the way the rest of the Alice environment is, they are introduced much later in the text, after students have developed a mastery of several other programming concepts.

We have found that interactive programs are fun and highly motivating to students. From a pedagogical perspective, however, Chapter 5 may be skipped. Some exercises and projects in later chapters use an interactive style, but you may be selective in examples and assignments.
Alice

The latest version of the Alice software and online galleries of 3D models can be downloaded from www.alice.org. A DVD is supplied with this book. The DVD contains two versions of Alice 2.2, one for PC and one for Mac. If you are using a PC with Linux, check the Web site www.alice.org for a version compatible with your system. The Alice Web site also provides instructions for installation, Frequently Asked Questions, and links for receiving bug reports. We maintain active debugging and support for Alice 2.2. Check www.alice.org for the latest releases.

The software included on the DVD is meant for individual use and is not adjusted for use in a networked lab environment. Visit www.alice.org download page for Alice 2.2, where you will find the latest instructions on how to install Alice 2.2 in a networked lab environment.

The Alice system is 3D graphics and memory intensive. The Alice development team has a set of minimum and recommended requirements for running Alice. Please note that many older laptops do not meet these requirements. It is extremely important to try Alice on the specific machines you will be using, just to be sure.

**Operating system requirements:**

*PC*

Windows® 7, Windows XP or Vista

*Mac®*

at least OS® X 10.4

**Minimum hardware requirements:**

A processor running at 500 MHz or better

VGA graphics card capable of high (16 bit) color

256 MB of RAM

Video resolution of 1024 × 768

A sound card

**Recommended hardware requirements:**

A processor running at 1.0 GHz or better

16 MB 3D video card (see www.alice.org for more details)

500 MB of RAM

Alice works well with digital projection systems for classroom demonstrations. Projectors limited to 800 × 600 video resolution will work, although 1024 × 768 is best.

Wanda Dann
Stephen Cooper
Randy Pausch