Preface to the Instructor

"…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 teaching introductory programming. There have been relatively few innovations in the teaching of programming in the last 30 years, despite the fact that introductory programming 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:

- 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, where the concept of an "object" is made tangible via on-screen objects that populate a three-dimensional 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 is 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 report) with an early introduction to events. In Alice, every object **is** an object that students can visibly see! We begin with objects in the very first chapter.

In our opinion, there are four primary obstacles to introductory programming:

1. **The fragile mechanics, particularly syntax, of program creation**
   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, as opposed to noticing that the "sum" variable has been decremented, rather than incremented. Alice's visual nature 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 the ability to share creations with peers; we have added the ability to run Alice programs in a WWW 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 under-represented student 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 know is an established movie-making process. We illustrate design techniques using simple sketches and screen captures. And, we encourage the use of textual storyboards, which are progressively refined, essentially designing with pseudocode.

How to use this text

Of course, as an instructor, you should use this text as you best see fit! Below are four ways we imagine the book being used, but you may discover additional ways:

1. **As the only text in a semester-long or short-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 a need for transitioning to a real-world language. Alternatively, this course can be used as a pre-CS1 course for students who would like to be, or are considering, a computing major but who 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 in CS1. The use of Alice in a pre-CS1 course has significantly reduced attrition for these students in our CS1 courses.

2. **As the first portion of a traditional "Introduction to Programming" course**, such as CS1. Both Seymour Papert's Logo and Rich Pattis' 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 are well 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 the Figure Preface-1.

3. **As the programming component of a "Computer Literacy" course**. At many schools, computer literacy courses attempt to give a broad introduction to computers and/or "information technology" to non-majors. 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.

4. **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 course could be a stand-alone course or as preparation for the College AP computing course.

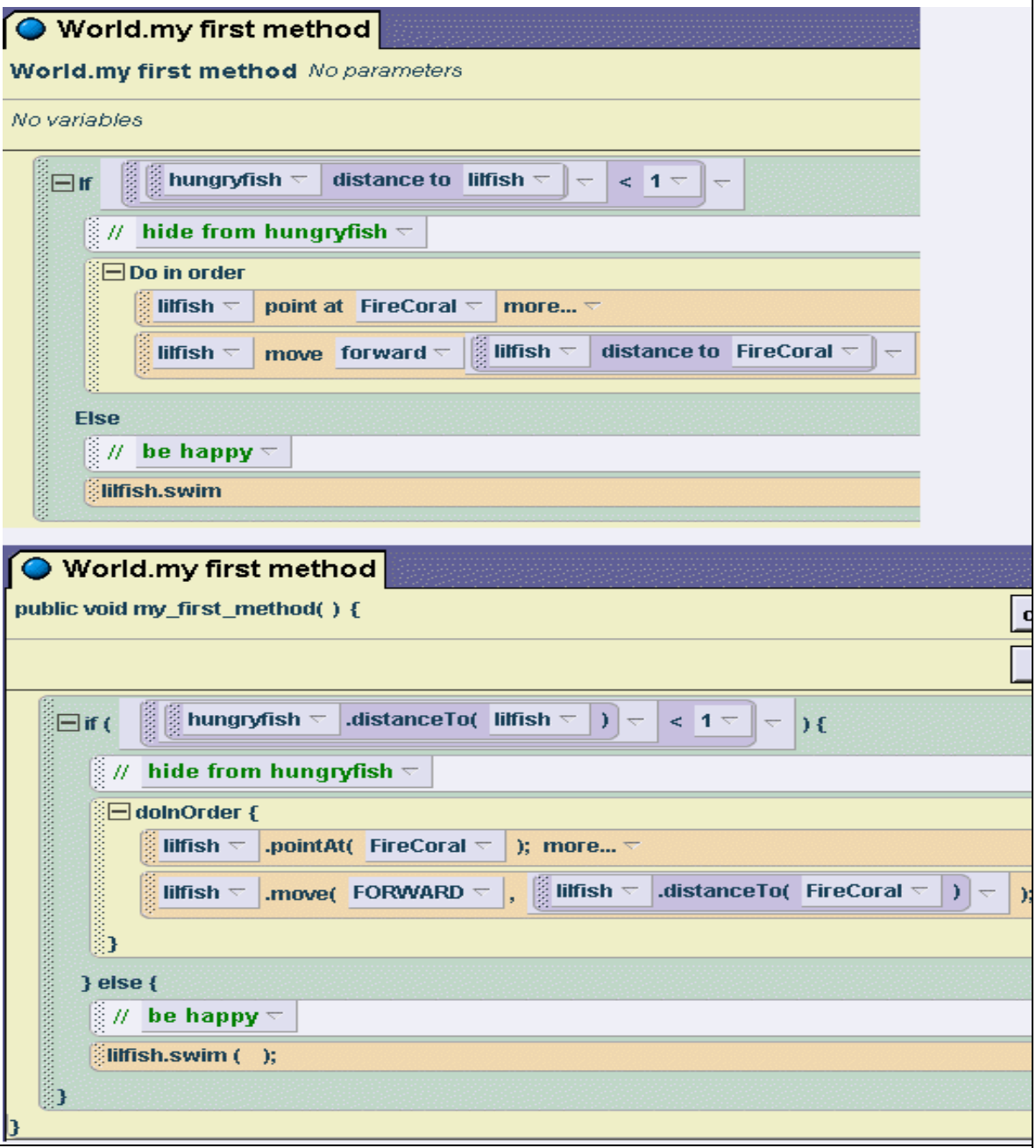


Figure Preface-1: Programs in Alice (top) can optionally be displayed with a Java-like syntax (bottom)

Structure of the book

The text begins with an introduction motivating why students would want to write a computer program and addressing any fears they may have about programming (especially helpful for computer literacy courses). We have a brief first chapter on basic Alice concepts. A traditional, paper-based tutorial is presented in Appendix A and the Alice system includes an interactive tutorial that some students may prefer.

Remaining chapters begin with overview and motivation of the chapter's topic and end with exercises, projects, and a summary. For your convenience, the following is a (very) brief overview of the major concepts covered, chapter-by-chapter. Clearly, the major focus of the text material is an introduction to the fundamental concepts of programming.

Chapter 1	Getting started
Chapter 2	Design, stepwise refinement, and a first program
Chapter 3	Built-in questions, expressions, and simple control statements
Chapter 4	Object Oriented programming: methods, parameters, and inheritance
Chapter 5	Interactive programs: events and event-handling
Chapter 6	Questions (functions) and conditional execution (If/Else)
Chapter 7	Definite (Loop) and indefinite loops (While) with methods
Chapter 8	Repetition: recursion
Chapter 9	Lists
Chapter 10	Variables, revisiting inheritance, and an array visualization
Chapter 11	A summary of fundamental concepts learned with Alice

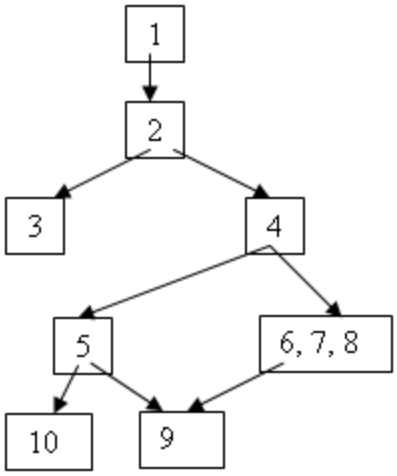
We recommend a rapid pace through the first 3 or 4 chapters of the book. (Chapters 1 and 2 can easily be covered in just 2 – 3 class days.) Instructors should be selective from the exercises at the end of each chapter. A large number of exercises are provided to allow the instructor to choose exercises most appropriate for their students. Examples of larger, more free-form "projects" are provided at the end of later chapters. We use the term "project" to label more advanced exercises that take more time than simple exercises.

Projects are meant to turn on the creative spirit, not weigh down the student. In particular, where feasible, we recommend "open-ended" projects. An open-ended project 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, uses a decision statement, and has 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 and Techniques" section. Collectively, the Tips & Techniques sections and Appendices A and B are a mini-User's Guide to Alice. The Tips & Technique sections 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 to text, laying the groundwork for using these techniques in programming examples that follow. Topic subheadings in **red** are used in text examples, topic subheadings in **green** are used in exercises. Other topics with subheadings in **black** are provided as a guide for those who want to learn more about animation with Alice. Appendix A is a "getting started" tutorial and Appendix B describes how to manage the Alice interface. Use of the interface is also integrated with text examples for programming concepts, where needed. These sections enrich the flavor of the book with selected "how to" topics.

Topic selection and sequence of coverage

Once again, as an instructor, the topic selection and sequence of coverage is in your hands! The dependency chart below should be helpful 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. In working with instructors in various college and university settings, we found topic selection and sequences are often based on time constraints, pedagogy, and philosophy of teaching. To provide a flexible framework for using this book, two basic paths through Chapters 1 - 4 were designed as:

Objects First: 1 – 2 – 4

Objects Early: 1 – 2 – 3 – 4

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 purposely 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 Chapters 7 and 8 can be reversed to allow coverage of recursion before *Loop* and *While* control structures.

Time-saving hints: If you are limited to 3 or 4 weeks for Alice (as part of a larger course), you may wish to assign exercises throughout with only 1 project, due at the end of the time block. Also, you can use Tips & Techniques sections as reading assignments, not requiring classroom presentation time. In our experience, students actually DO read the Tips & Techniques sections!

Notes concerning specific aspects of the text

If you are using this text to teach/learn Alice without discussing design, you may skip section 1 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. 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 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 pedagogic perspective, however, Chapter 5 may be skipped. Some exercises and projects in later chapters use an interactive style, but you may be selective in what you use as examples and assignments.

A note on running Alice

The version of Alice on the disk supplied with this book is meant to run on a PC running Windows ME, 2000, or XP operating system. If you are using a Macintosh or a PC with Linux, check the web site http://www.alice.org for a version of Alice for your operating system.

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 it on, just to be sure.

Minimum hardware requirements:
- A Pentium running at 500 MHz or better
- A VGA graphics card capable of high (16 bit) color
- 128 MB of RAM
- Video resolution of 1024x768

Recommended hardware requirements:
- A Pentium running at 1.0 GHz or better
- 16 MB 3D video card (see www.alice.org for more details)
- 256 MB of RAM
- A sound card

Alice works well with digital projection systems for classroom demonstrations. Projectors limited to 800x600 video resolution will work, although 1024x768 is best.

Acknowledgements

As noted above, Seymour Pappert's Logo and Rich Pattis' Karel the Robot were great inspirations in using a visible micro world. Alan Kay and the Squeak team inspired us to create the mouse-based program editor, and we were also inspired by the syntax-directed editor work done by Tim Teitelbaum. We are indebted to George Polya, Mike Clancy, and Doug Cooper for our problem-solving approach.

Our deep gratitude goes to early testers and users of our text and instructional materials for their helpful comments and suggestions: Susan Rodger (Duke University), Rick Zaccone (Bucknell University), Bill Taffe (Plymouth State), Angela Shifflet (Wofford College), and William Taylor

(Camden County College). In addition, we are thankful for the assistance of our students: Toby Dragon (Ithaca College), Kevin Dietzler (Saint Joseph's University), Patricia Hasson (Saint Joseph's University), and Kathleen Ryan (Saint Joseph's University).

The life and breath of the Alice software is dependent on a group of creative, energetic, and dedicated graduate students, undergraduate students, and staff members group at Carnegie Mellon University. Without these people, Alice does not live and we could not have written this textbook. The primary authors of this version of Alice include Ben Buchwald, Dennis Cosgrove, Dave Culyba, Cliff Forlines, Jason Pratt, and Caitlin Kelleher, but a more complete list is available at www.alice.org. Many artists at Carnegie Mellon have graciously placed their work into the gallery for the benefit of others; we list Sarah Hatton, Mo Mahler, Shawn Lawson and Tiffany Pomarico here, but the contributors run into the hundreds. Tommy Burnette, Kevin Christiansen, Rob Deline, Matt Conway, and Rich Gossweiller all made seminal contributions to earlier versions of Alice at the University of Virginia, who we also thank for its support and encouragement of earlier versions of Alice.

We would like to thank Alan Apt and Prentice Hall for supporting this effort.

Over the last ten years, the National Science Foundation, DARPA, NASA, Apple, Ford, Intel, Microsoft Research, and SAIC have contributed support for the development of the Alice system, for which we are most grateful.