STUDENT PERFORMANCE IN COMPUTER MODELING AND PROBLEM SOLVING IN A MODERN INTRODUCTORY PHYSICS COURSE

by

Matthew Adam Kohlmyer

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the field of

Physics

Carnegie Mellon University
Mellon College of Science
Department of Physics

Pittsburgh, PA

2005
ABSTRACT

KOHLMYER, MATTHEW ADAM. Student Performance in Computer Modeling and Problem Solving in a Modern Introductory Physics Course. (Under the direction of Ruth W. Chabay.)

*Matter & Interactions*, an innovative introductory physics curriculum developed by Ruth Chabay and Bruce Sherwood, emphasizes computer modeling and fundamental physical principles. Two think-aloud protocol studies were conducted to investigate the performance of students from this curriculum in solving physics problems that require computer modeling. Experiment 1 examined whether *Matter & Interactions* students would, given the choice, use computer modeling to solve difficult problems that required predicting motion, and how their solution approaches differed from those of students from a traditional introductory physics course. Though they did not overwhelmingly choose computer modeling, some *M&I* students did write computer models successfully or apply the iterative algorithm by hand. The solution approaches of *M&I* students and traditional course students differed qualitatively in their use of the momentum principle and pre-derived special case formulas.

In experiment 2, *Matter & Interactions* students were observed while they wrote programs in the VPython language in order to examine their difficulties with computer modeling. Areas of difficulty included determining initial conditions, distinguishing between simulated time and the time step, and updating momentum and position. Especially troublesome for students was the multistep procedure for calculating a force that changes with time. Students' understanding of the structure of a computer model improved by the end of the semester as shown by their performance on a line sorting task. Students with fewer difficulties proceeded through the computer model in a more linear, straightforward fashion. Instruction was revised based on initial findings from the first phase of the experiment. Students in the second phase of the experiment, who had used the revised instruction, had fewer difficulties on the same tasks, though other factors may have been involved in the improvement.
ACKNOWLEDGEMENTS

Many people were instrumental in the development and completion of this thesis. I first wish to thank the Department of Physics of Carnegie Mellon University for its willingness to support an educational research project. I also thank the National Science Foundation, who provided partial funding for this project (NSF grant #DUE-0320608).

Two "surrogate families" welcomed me as one of their own during the years I spent working on this thesis. The first was the (now defunct) Center for Innovation in Learning at Carnegie Mellon. I thank CIL's former faculty, postdocs, students, and staff for their support and guidance, especially Dick Hayes, Fred Reif, and Marsha Lovett.

My second "surrogate family" was the Physics Education Research Group at North Carolina State University. Special thanks to Robert Beichner and John Risley for their advice, feedback, and support. Thanks also to former and current students and postdocs, including David Abbott, Jeanne Morse, Sejung Kim, Lin Ding, and Vince Kuo, for many useful conversations, and for their friendship.

I owe a tremendous debt of gratitude to my thesis advisors, Ruth Chabay and Bruce Sherwood. Their contributions to this research have been immeasurable, and their guidance and friendship have been invaluable. From them I have learned so much about physics, computation, teaching and learning, research, and life in general. The opportunity to work with them has truly been a privilege.

Lastly, I thank my parents, Joan and Paul Kohlmyer, and my brother, Steve Kohlmyer, for their unwavering love and encouragement.
# TABLE OF CONTENTS

List of Tables.....................................................................................................................vii

List of Figures...................................................................................................................viii

1 Introduction................................................................................................................. 1

1.1 *Matter & Interactions* .......................................................................................... 1

1.2 Computer modeling in introductory physics....................................................... 2

1.3 Fundamental principles and problem-solving..................................................... 6

1.4 Research questions.............................................................................................. 7

1.5 Research methods ............................................................................................... 9

2 Background............................................................................................................... 11

2.1 Computer modeling .......................................................................................... 11

2.1.1 M.U.P.P.E.T............................................................................................... 12

2.1.2 STELLA..................................................................................................... 16

2.1.3 Boxer........................................................................................................ 21

2.2 Expertise in solving physics problems.............................................................. 28

2.3 Research methods involving verbal data .......................................................... 34

2.3.1 "Think-aloud" protocol analysis ................................................................. 34

2.3.2 Demonstration interviews .......................................................................... 36

2.3.3 Tutoring sessions ....................................................................................... 37

2.3.4 Verbal analysis............................................................................................ 38

3 Experiment 1: The "hard problems" study............................................................. 42

3.1 Pilot study .......................................................................................................... 42

3.1.1 Pilot study: participants ............................................................................ 43
3.1.2 Pilot study: procedure ............................................................................... 43
3.1.3 Results of pilot .......................................................................................... 44
3.2 Participants........................................................................................................ 49
3.3 Procedure .......................................................................................................... 53
3.4 Analysis............................................................................................................. 55
3.5 Use of computation, iteration, and integration.................................................. 56
  3.5.1 Computation and iteration methods in the M&I group............................. 57
  3.5.2 Integration use in the traditional group..................................................... 68
3.6 Differences in solution approaches................................................................... 76
  3.6.1 Use of momentum methods ................................................................. 79
  3.6.2 Special case formulas: circular motion.................................................... 89
  3.6.3 Special case formulas: constant acceleration kinematics.......................... 96
  3.6.4 Textbook usage ....................................................................................... 102
  3.6.5 Problem completion................................................................................ 103
3.7 Discussion....................................................................................................... 104
4 Experiment 2: The VPython programming study ............................................... 108
  4.1 Participants..................................................................................................... 108
  4.2 Experimental procedure.................................................................................. 110
  4.3 Instructional approaches ................................................................................. 111
    4.3.1 Computer modeling instruction in Spring 2003................................. 112
    4.3.2 Computer modeling instruction in Fall 2003...................................... 113
  4.4 Nature of the interviews.................................................................................. 117
  4.5 Qualitative discussion of student difficulties............................................... 122
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Date</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6</td>
<td>Appendix 6: 3-D classroom model computer activity, Spring 2003</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>7.7</td>
<td>Appendix 7: 3-D Model of Motion activity, Spring 2003</td>
<td></td>
<td>231</td>
</tr>
<tr>
<td>7.8</td>
<td>Appendix 8: Vector worksheet, Fall 2003</td>
<td></td>
<td>233</td>
</tr>
<tr>
<td>7.9</td>
<td>Appendix 9: Intro to VPython tutorial, Fall 2003</td>
<td></td>
<td>237</td>
</tr>
<tr>
<td>7.10</td>
<td>Appendix 10: Modeling motion tutorial, Fall 2003</td>
<td></td>
<td>245</td>
</tr>
</tbody>
</table>
LIST OF TABLES

Table 3.1. Participants from Traditional course............................................................... 49
Table 3.2. Participants from M&I course......................................................................... 49
Table 3.3. Number of solution attempts where computation, iteration, and integration were invoked................................................................................................... 57
Table 3.4. Number of solution attempts with specific features, by group. ......................78
Table 3.5. Number of solution attempts with specific features, by problem......................79
Table 4.1. Volunteers for Experiment 2 and the final grades they received in their introductory physics courses................................................................. 109
Table 4.2. Results for phase 1 (first semester), session 1, table and stools program........ 157
Table 4.3. Results for phase 2 (second semester), session 2, Jovian moons program ... 158
Table 4.4. Results for phase 1, session 1, moving rock program .................................... 161
Table 4.5. Results for phase 2, session 1, moving rock program .................................... 161
Table 4.6. Results of coding by intervention type, phase 1 (first semester), session 2 (orbit program).............................................................................................. 165
Table 4.7. Results of coding by intervention type, phase 2 (second semester), session 2 (orbit program).............................................................................................. 165
Table 4.8. Results of coding by topic of intervention, phase 1, session 2 ....................168
Table 4.9. Results of coding by topic of intervention, phase 2, session 2 .....................168
Table 4.10. Results of session 3 sorting task, phase 1 ...................................................173
Table 4.11. Results of session 3 sorting task, phase 2 ...................................................173
LIST OF FIGURES

Figure 2.1. Example of a M.U.P.P.E.T Pascal program to model the 1-D motion of projectile in the vertical direction, with air resistance. Adapted from Redish & Wilson, 1993, available at http://www.physics.umd.edu/perg/papers/redish/mupajp.html ............. 14

Figure 2.2. Graphical output of the program shown in figure 2.1. From Redish & Wilson, 1993, available at http://www.physics.umd.edu/perg/papers/redish/mupajp.html ............. 15

Figure 2.3. A STELLA model of projectile motion. The upper window contains the icon-based model. The lower window shows the trajectory in x-y coordinates. Created with STELLA 8, High Performance Systems, Inc. See http://www.hps-inc.com ........................................................................ 18

Figure 2.4. Equations generated by the projectile motion model in Figure 2.1 .......... 19

Figure 2.5. Boxer program that models the motion of a projectile near the surface of the earth (constant downward acceleration, no air resistance). Created with PyxiBoxer by Andrea A. diSessa and Edward H. Lay, PyxiSystems LLC. See http://www.pyxisystems.com ........................................................................ 23

Figure 3.1. Problem 1: satellite orbiting earth. ......................................................... 54

Figure 3.2. Problem 3: block on a spring, moving in two dimensions. ....................... 54

Figure 3.3. Martin, problem 1 .................................................................................. 60

Figure 3.4. Martin, problem 1 .................................................................................. 61

Figure 3.5. Carter, problem 1 ................................................................................... 63

Figure 3.6. Zach, problem 2 ..................................................................................... 64

Figure 3.7. Zach, problem 2. Note updating of velocity at top ................................... 65

Figure 3.8. Josh, problem 2 ....................................................................................... 70

Figure 3.9. Josh, problem 2. Work "in x direction" is the force times the distance ....... 70

Figure 3.10. Jerome, problem 2 ................................................................................. 72

Figure 3.11. Jerome, problem 2. Note "x" and "y directions" for work. ......................... 72

Figure 3.12. Jerome, problem 2. Using work-kinetic energy theorem ......................... 73

Figure 3.13. Thomas, problem 2 .............................................................................. 74
Figure 3.14. Donald, problem 1. ......................................................................................75

Figure 3.15. Donald, problem 1. Note integral of gravitational force and an impulse
calculation in lower right. .......................................................................................... 76

Figure 3.16. Zach, problem 1. ........................................................................................ 84

Figure 3.17. Zach, problem 1. Change in $v$ in lower right used to find new $v$ ............ 84

Figure 3.18. Zach, problem 1. Note energy principle, below diagram. ......................... 85

Figure 3.19. Zach, problem 1. Note momentum principle in lower left. ....................... 85

Figure 3.20. Donald, problem 2. ......................................................................................87

Figure 3.21. Donald, problem 2. Calculation of a new $x$-component of $v$ in lower right.87

Figure 3.22. Donald, problem 2. Calculation of a new $y$-component of $v$ .................... 88

Figure 3.23. Donald, problem 2. Using constant acceleration kinematics to find a
distance. ...................................................................................................................... 88

Figure 3.24. Jerome, problem 1. ..................................................................................... 91

Figure 3.25. Jerome, problem 1. Notice circumference calculation at bottom, distance
calculation at top right, and ratio of distance to circumference at lower right.  
The answer says, ".628 of way around its orbit." .................................................... 92

Figure 3.26. Sam, problem 3. "$x_0$" is the original spring length, "$x$" is the stretch ...... 93

Figure 3.27. Sam, problem 3. .......................................................................................... 93

Figure 3.28. Thomas, problem 1. .................................................................................... 94

Figure 3.29. Thomas, problem 1. Note his alteration to the path of the satellite ............ 95

Figure 3.30. Denise, problem 3. ...................................................................................... 96

Figure 3.31. Eleanor, problem 2. ..................................................................................... 98

Figure 3.32. Denise, problem 1. ...................................................................................... 100

Figure 3.33. Denise, problem 1. Calculating an acceleration. ........................................ 101

Figure 3.34. Denise, problem 1. Note use of "1/2 $at^2$" formula. ................................... 102

Figure 4.1. Diagram from the instructional materials designed and used for the second
semester. It is one of example of explicit mapping of algebraic notation to
VPython syntax. ........................................................................................................ 116
Figure 4.2. Example of dialogue from session 2, moon orbit program. ...................... 121
Figure 4.3. Bobby's program before correcting structural errors............................ 125
Figure 4.4. Using time and time step in a VPython computer model.......................... 130
Figure 4.5. Example of the procedure used to calculate the gravitational force vector. 139
Figure 4.6. Major interventions in the first program (arrows and position vectors) from
session 1. Phase 1 results are in black, phase 2 results are gray. ...................... 158
Figure 4.7. Comparison of the mean number of interventions, session 1, first program
(arrows and position vectors). ........................................................................... 159
Figure 4.8. Total interventions versus completion time for program 1, session 1........... 160
Figure 4.9. Major interventions in the moving rock program from session 1. Phase 1
results are in black, phase 2 results are gray. .............................................. 162
Figure 4.10. Comparison of the mean number of interventions, session 1, moving rock
program. .......................................................................................................... 163
Figure 4.11. Total interventions versus completion time for program 2, phase 1 (moving
rock). ................................................................................................................. 164
Figure 4.12. Major interventions in the orbit program. Phase 1 student data are in black,
phase 2 results are in gray. ........................................................................... 166
Figure 4.13. Comparison between the two phases of the mean number of types of
interventions, session 2 .................................................................................... 167
Figure 4.14. Total interventions versus completion time for session 2 (moon orbit
program). .......................................................................................................... 167
Figure 4.15. Mean number of interventions by topic, session 2 ................................. 169
Figure 4.16. Randomly arranged lines of a program, used in the sorting task in session 3.
Students copied and pasted the lines into the "skeleton" of a program, shown
in figure 4.17 ................................................................................................. 171
Figure 4.17. The outline of VPython program from session 3. Students copied and
pasted randomly arranged chunks of line, seen in figure 4.16, into this
"skeleton" to make a working computer model. ............................................. 172
Figure 4.18. Bobby (phase 1): session 2 intervention category plot........................... 176
Figure 4.19. Norman (phase 1): session 2 intervention category plot ....................... 176
Figure 4.20. Paul (phase 1): session 2 intervention category plot. ................................177
Figure 4.21. Richard (phase 1): session 2 intervention category plot...........................177
Figure 4.22. Andrew (phase 2): session 2 intervention category plot. .......................... 178
Figure 4.23. Charles (phase 2): session 2 intervention category plot. ...........................178
Figure 4.24. David (phase 2): session 2 intervention category plot............................... 179
Figure 4.25. Kyle (phase 2): session 2 intervention category plot................................. 179
Figure 4.26. Nick (phase 2): session 2 intervention category plot................................. 180
Figure 4.27. Number of transitions between intervention categories for each student.  
Phase 1 student data are in black, phase 2 results are in gray.................................... 180
Figure 4.28. Number of transitions back to previous intervention categories for each  
student. Phase 1 student data are in black, phase 2 results are in gray.............. 181
Figure 7.1. Comparison of the satellite's trajectory with and without the 400 N force for  
10 minutes in problem 1. The above program displays the "with rocket"  
trajectory. The trail has been added for clarity............................................. 206
Figure 7.2. Graphical output of the above program, with the addition of a trail. The  
position of the ball in the case of negligible air resistance differs by only a  
few centimeters............................................................................................. 208
Figure 7.3. Graphical output of the above program, with the addition of a trail. The  
sphere represents the block on the spring that is attached to the peg at the  
origin.............................................................................................................. 210
Figure 7.4. Sample graphical output from program 1, session 1, phase 1 only (table and  
stools). ......................................................................................................... 214
Figure 7.5. Sample graphical output from program 1, session 1, phase 2 only (static  
Jovian moons).............................................................................................. 216
Figure 7.6. Sample graphical output for program 2, session 1 (moving rock). The rock  
(dark sphere) is the same size as the lighter sphere representing the origin,  
but has moved past the origin in the –z direction........................................ 218
Figure 7.7. Sample graphical output of program 3 (moon orbit). A trail has been added  
for clarity........................................................................................................ 220
1 Introduction

The purpose of this dissertation is to investigate the performance of students from a modern, innovative introductory physics curriculum on challenging problems, especially those that call for the use of computer modeling or iterative solution methods. The emphasis on these methods distinguishes this curriculum from the traditional introductory physics course. Before outlining the research questions and methods, I will first briefly introduce this curriculum, give the motivations behind its development, and describe its main differences from the typical introductory physics course.

1.1 Matter & Interactions

Over the past 25 years, research in physics education has shown in many instances that students in the traditional university-level introductory physics course fail to learn even the most basic physics concepts (see, for example, McCloskey, Carramazza, and Green, 1980; Halloun and Hestenes, 1985). Many research-based instructional innovations have emerged to improve student understanding through interactive pedagogy. However, even among these innovative curricula, the content of the introductory physics course has, for the most part, remained unchanged over the past century. Most introductory mechanics courses and textbooks tend to follow the same sequence of topics (kinematics, Newton's laws, energy, then momentum), and they typically do not emphasize the modeling process that physicists use when analyzing a system. They tend to focus on algebraic problem solving, often through use of special case formulas. For example, in kinematics, constant acceleration motion is a major topic,
and students often solve these problems by plugging parameters into pre-derived constant acceleration formulas.

*Matter & Interactions* (Chabay and Sherwood, 2002), developed by Ruth Chabay and Bruce Sherwood, is an innovative introductory physics curriculum that was designed to bring modern content into the introductory course. It focuses on the atomic nature of matter and emphasizes physical modeling. By physical modeling, I mean the process of constructing an abstract description of a real physical system that can be used to predict or explain the behavior of the real system. The behavior of the abstract model is governed by fundamental physical principles, while judicious use of assumptions and approximations simplifies the model to make it understandable, yet still powerful. The model can be refined to make more accurate predictions, or to explain more details of the behavior of the real system, but at the cost of more complexity. Students in *Matter & Interactions* (hereafter referred to as M&I) practice this physical modeling approach in several ways. One is through solving problems that are more complicated than the typical end-of-chapter physics problem. Another is through using a computer to model complicated systems.

### 1.2 Computer modeling in introductory physics

Modern research in physics consists of three main areas: theory, experiment (or observation), and computation. Theoretical and experimental research have dominated physics for hundreds of years, while computational physics has become prominent since the emergence of the digital computer in the 20th century. Computers are used not only to analyze data from experiments or to solve mathematical equations, but also to simulate or
model systems. Computer modeling enables researchers to perform "experiments" on systems that would be impossible or too dangerous to investigate in the laboratory (e.g. astrophysical systems); to explore the behavior of complicated systems based on theoretical principles; and to attack problems that may be analytically intractable.

While computer modeling is an important technique in modern physics research, it is typically neglected in undergraduate physics education, especially in the introductory course. In the typical traditional introductory physics course, mathematical formalism is often the sole focus of lectures and problem-solving assignments, and lab sections focus on experimental techniques. If one of the goals of an introductory physics course is to expose students to the important aspects of physics, then one could argue that computer modeling should be a component of such a course, along with theory and experiment. Computer modeling in an introductory course could be used as a way for students to make connections between theory and experiment and offer students an additional way to apply physical principles beyond the usual end-of-chapter problems. M&I has incorporated computer modeling with such goals in mind.

In particular, a computer model could make clear to students the power and primacy of fundamental physical principles. For example, to model a mechanical system of interacting particles, one could create a computer model based on the following algorithm:

1. Declare the initial positions and momenta of the particles;
2. Based on the appropriate force law and the superposition principle, calculate the net vector force on each particle;
3. Using the momentum principle (Newton's 2nd law), calculate the new
momentum of each particle a short time later, based on these forces;

4. Using the definition of velocity, calculate the new position of each particle, based on the new momentums;

5. Repeat steps 2 through 4 for as many time-steps as desired

This is a simple way (Euler's Method) to numerically integrate the differential equations of motion. This same core algorithm could be used to model the behavior of a wide variety of situations (e.g. planetary orbits, harmonic oscillation, Rutherford scattering).

Computer modeling can also help emphasize the time evolutionary character of Newton's second law. Typically in traditional introductory physics courses, Newton's second law is presented in the form \( \mathbf{F}_{\text{net}} = m\mathbf{a} \), which suggests only an algebraic relationship between force and acceleration. The types of problems students solve in the traditional course tend to reinforce this idea; typically, Newton's second law is used to find unknown forces or accelerations in systems where the motion known or constrained. The algorithm used in computer modeling could help students to understand that net force produces a change in momentum (or velocity) over a time period. Momentum, and therefore position, could be predicted arbitrarily far into the future provided the force is known at each time step.

There are many different types of software that can be used to model physical systems with a computer, such as spreadsheets, symbolic algebra packages, or simulation-building software (e.g. Interactive Physics), but another choice is write a program in a high-level programming language. Such a program would contain an algorithm such as the one described above. One advantage to modeling with a programming language is that the required physical principles are written explicitly (in
the appropriate syntax), unlike a pre-built simulation or a pre-rendered video. The physics is not "hidden," and so students can more easily make connections between theoretical principles and the behavior of systems. Disadvantages to programming include the need to learn arcane syntax, and the difficulties in making graphic visualizations with a programming language.

One particular programming environment, VPython, has been designed with the goal of reducing difficulties of both syntax and graphics creation (see http://vpython.org). The M&I course as it is taught at a number of different institutions, including Carnegie Mellon University, Purdue University, and North Carolina State University, uses VPython for computer modeling exercises. VPython has two major components. The first is the Python programming language, an interpreted, object-oriented language created by Guido van Rossum (see http://python.org). Python features clear syntax and is relatively easy to learn, making it suitable for introductory students. The second main component is Visual, a module created by David Scherer that allows easy creation of 3D graphics. The user only has to specify the types of graphical objects and their geometric attributes; when the program is run, a graphical window that displays the stated objects is created automatically. A separate thread runs in the background, independent of the user, that constantly monitors object positions, sizes, orientations, and colors, and renders a 3D graphical image based on their current values in the running program. The 3D animation in the graphical window is therefore a side effect of the physics computations in the user's code; the user does not have to worry about specific commands for screen management.
1.3 Fundamental principles and problem-solving

Physics is a reductionist science; that is, it tends to reduce the number of observed relationships and experimental results to a small number of principles, which can predict and explain a wide variety of complicated phenomena. For example, in classical mechanics, there are only three fundamental dynamical principles: the momentum principle, the energy principle, and the angular momentum principle. Typically in introductory physics courses, however, the importance of fundamental principles is not explicitly communicated, and the tasks that students perform in these courses may obscure the fact that some equations are more fundamental than others. Problems that students solve in traditional courses may all be of a similar type or follow a similar script, encouraging students to "pattern-match" the solution to a known example or archetype, without giving any thought to the physical principles invoked in the example. Also, the limited scope of problems may encourage students to rely on special-case formulae (such as equations for motion during constant acceleration), thus unduly heightening the importance of these formulae in the minds of students and leading students to use them in inappropriate situations.

In contrast, the types of problems that students solve in M&I are designed to promote student use of fundamental principles and discourage "problem-matching." Part of the process of physical modeling that M&I describes is to begin an analysis of a system with fundamental principles, and instructors make this explicit to students. Students in this course often solve problems that are more complicated than the usual "end-of-chapter" problems, ones that cannot be solved by simply "plugging" numbers into a single formula. Instead, students must choose an appropriate general principle, and
then derive any specific formulas using the quantities from the physical system that is being analyzed. Therefore, because students in M&I practice solving problems that require the use of general principles, they may learn to use this approach when faced with new problems.

1.4 Research questions

The use of Matter & Interactions in the introductory course opens up a host of new research issues. Because the curriculum is so different from the typical introductory physics course, little is known about what difficulties students have with this curriculum, how it affects students' problem solving abilities, and how it can most effectively be taught.

In this dissertation I will focus primarily on student use of computer modeling in the M&I course. Originally, I was interested in examining whether students from the M&I course would choose to apply computer modeling or iterative procedures to solve new, challenging problems that involve predicting motion. Because students in M&I solve a variety of problems, both analytically and numerically using VPython, it is possible that they learn a wide variety of problem-solving methods. Students may see computer modeling as another problem-solving tool to use, in addition to algebraic methods, when faced with an unfamiliar problem. To examine this issue, a study was conducted, described in chapter 3, where M&I students were asked to solve difficult problems using any means, analytic or iterative, that they saw fit.

In the course of running this study, a secondary research interest emerged. Even if M&I students do not use computer modeling or iterative techniques when faced with a
new problem, we might be able to learn what equations, principles, or methods they consider fundamental. *M&I*'s emphasis on fundamental principles may lead students to approach unfamiliar or challenging problems differently from students in a traditional course. By comparing the approaches of *M&I* students on these hard problems to the approaches of traditional students, we might be able to see how each course affects students' choice of principle and tendency to "problem-match" or use inappropriate special-case formulas. Therefore, the two main research questions that this study addresses are the following:

1. Given the emphasis placed on computer modeling in the *M&I* course, will students from this course use the computer, or an iterative application of the momentum principle, when faced with a new or challenging problem?
2. What differences are there between students in the *M&I* course and students from a traditional introductory physics course in their approaches to new or challenging problems with regard to use of fundamental principles and "problem matching"?

The results of this experiment opened up a number of research areas related to *M&I* that could be pursued further. One main area is how the *M&I* curriculum is affecting students' problem solving strategies. This includes the procedural and conceptual resources they use when solving problems, the heuristics they use when deciding what principles or equations to implement, and their ability to evaluate their solutions.

Another area of interest is students' capabilities and difficulties with computer modeling. I chose to examine this topic in a second study. This is an important topic for
both pure research and for educational practice. The details of teaching computer modeling loom large in *M&I*. Despite the ease-of-use advantages of VPython, programming is not a trivial skill to learn, and if students are going to be expected to write computer models, instructors must invest time and effort into teaching the requisite particulars of the language. Learning the syntax of the language is not enough; students must also learn how to structure their programs and how to translate the necessary principles into computer code. There is limited time available to teach programming in a physics course, which makes it even more important for *M&I* instructors to know what students find difficult about computer modeling, and what instruction will help alleviate these difficulties. These issues lead to the following two research questions that were investigated in the study that will be discussed in chapter 4:

3. What difficulties do students in the *M&I* course have with computer modeling?
4. What instructional interventions can help alleviate these difficulties?

### 1.5Research methods

The experiments that I performed while investigating these questions involved observation of students solving problems, both through handwritten solutions and through computer modeling. Although data from students' exams, quizzes, and homeworks in the *Matter & Interactions* can reveal certain areas of difficulty, students' finished solutions (or programs) often do not reveal the process that was used to solve the problem. Observation of students performing a problem solving task, along with analysis of what they say when working on the task, can give rich, detailed data of the process,
including what steps students take in their solutions, where they made mistakes or false starts, and what aspects of the problem are difficult for students to grasp. Students' solutions were videotaped, providing a visual record of the solution path. I will go into more detail on the relevant research methods in section 2.3.
2 Background

In this chapter I will provide background in three areas related to this study:

1. previous attempts to implement computer programming in introductory physics courses, including their relevant educational results;
2. some past research into students' abilities to solve problems in introductory physics courses, and;
3. the research methods and analyses that were employed in this study.

2.1 Computer modeling

Many attempts have been made to use computer programming in the introductory university physics curriculum. Using an iterative application of the momentum principle to predict the motion of objects is discussed in Volume I of The Feynman Lectures on Physics (Feynman, Leighton, & Sands, 1965). Here, the numerical procedure that is essentially the same as the one I outlined in section 1.3 is used to calculate the motion of a harmonic oscillator and a planet in orbit. The possibility of using "a machine to handle the arithmetic" is mentioned, but details are not given, and the iterative procedure is not used in the text again. As early computers became available at universities in the late 1960's, some physics instructors began to teach students how to program a computer to do the repetitive calculations of this numerical procedure (see, for example Bork, 1967). Output was usually printed numbers on hardcopy, since these early computers typically lacked any means of visualization, such as a CRT monitor screen. It wasn't until the late 1970's and 1980's that small, relatively cheap, modern microcomputers, complete with monitors, became more widely available. It was in this era that experimentation with
computers in all classrooms, not just physics classrooms, emerged.

Below, I will review a few of the major attempts to use computer modeling in introductory level physics instruction. In many cases where computer modeling was introduced to physics instruction, it was simply a sequence added on to an existing course that left no major impact on course content or student learning. For this reason, I will concentrate on efforts that, like *Matter & Interactions*, involved not only the introduction of computer modeling to physics instruction, but also a restructuring of course goals, course content, or instructional approach. Furthermore, I will focus on curricula where students were involved in constructing the computer models from first principles, and not just using pre-existing simulations or "microworlds" to explore Newtonian concepts.

2.1.1 *M.U.P.P.E.T.*

In the 1980's, the University of Maryland physics department began a computer-based educational effort called M.U.P.P.E.T.: the Maryland University Project in Physics and Educational Technology. M.U.P.P.E.T.'s goal was to introduce computing into the undergraduate physics curriculum, thereby enabling instructors to broaden and reorganize course content to reflect contemporary physics, to train students through visual simulations, and to provide students with experience in analyzing real, complex physical systems early in their training. (MacDonald, Redish & Wilson, 1988). The design principles for M.U.P.P.E.T. were that computers should be used to teach material that could not otherwise be taught before; that they should complement, not replace, the existing course elements (e.g. text, instructor, laboratory); and that students must be actively engaged in controlling the computer, not being led by the computer. These principles are similar to themes that occur in *Matter & Interactions*. 

12
Students in the M.U.P.P.E.T. courses were taught a subset of the computer language Turbo Pascal; it was deemed easiest to learn among common high-level computer languages, and it was believed that it would provide students with skills that would be transferable to other professional languages, like C or FORTRAN, that they might encounter later in their careers (Redish and Wilson, 1993). Pascal is more complicated than VPython; it requires explicit declaration of all variables used in a program, and it does not automatically generate graphics as VPython does. The programs that students in M.U.P.P.E.T. courses wrote were often built up from pre-written procedures. Figure 2.1 shows an example of a Pascal program that models the motion of a particle in one dimension, such as a ball thrown vertically into the air, using the forces of gravity and air resistance. The given procedures in the program include one to create a screen that contained the needed initial conditions (MakeDataScreen), one to take the data from the screen to use in the mathematical calculations (GetScreenData), others to create the two-dimensional graphical output (GraphSetUp, PlotIt), and so on. (The graphical display generated by this program is shown in figure 2.2.) The only part of the program that students would have to write themselves was the iterative numerical integration of Newton's 2nd law; note the comment, "Student to fill in this procedure" in the "StepEuler" procedure. So, just as in M&I, the idea of using an iterative application of a fundamental physics principle to predict the motion of complicated systems was a main focus of the course, although M.U.P.P.E.T.-based course curriculum was not as radically altered from traditional physics curriculum as was M&I.
Figure 2.1. Example of a M.U.P.P.E.T Pascal program to model the 1-D motion of projectile in the vertical direction, with air resistance. Adapted from Redish & Wilson, 1993, available at http://www.physics.umd.edu/perg/papers/redish/mupajp.html
The literature on M.U.P.P.E.T. focuses on how it was used in the classroom, the types of new, complex physics problems it opened up for student analysis, and how it altered the material that could be covered in the introductory course. (For example, new material such as nonlinear dynamics could now be covered, but at the expense of rigid body motion and fluid dynamics.) Less is said about what difficulties students had with programming or with the new types of problems, and whether any modifications needed to be made to alleviate student difficulties. Johnston and McPhedran (1993) discuss the results of using M.U.P.P.E.T. materials with second and third year students. These students, who volunteered to study M.U.P.P.E.T. materials, performed better than the
traditional students in traditional tests in certain subject areas. However, the results are not very relevant for comparing to \textit{M&I}, since the computer modeling problems extended into advanced topics such as solid state physics, plasma physics, and Fourier transforms.

Redish and Wilson report that the use of M.U.P.P.E.T. enabled instructors to assign independent research projects to students, something that is rare in the traditional introductory course. About two thirds of the students performed projects that were "valuable and interesting." Redish and Wilson believe that computer modeling was integral to the success of these projects, since an earlier attempt at assigning projects, before the use of M.U.P.P.E.T., resulted in few projects that had "any characteristics of normal scientific research" (Redish and Wilson, 1993). These projects also led to increased student interest in continuing independent research later in students' academic careers.

2.1.2 \textit{STELLA}

Niedderer and colleagues at the Institute of Physics Education, University of Bremen, Germany, have also introduced computer modeling in physics instruction (Niedderer, Schecker, and Bethge, 1991). Their goals and design principles were similar to those of Maryland's M.U.P.P.E.T. program; for example, they wished to give students the opportunity to tackle more complicated systems than would otherwise be possible using algebraic mathematics alone (Schecker, 1993b). However, they felt that traditional programming languages were too difficult to learn and placed too much emphasis on symbolic equations. Instead, they use a graphical modeling software package called STELLA (from High Performance Systems, Inc.).
In STELLA, all quantities are represented as graphical icons, and users construct models by linking the icons together in ways that represent their numerical relationships. Box icons, called stocks, indicate quantities that can be summed up or integrated, such as an object's position. Another icon, called a flow, represents a rate of change of a summed quantity. For example, the upper window in figure 2.3 is a simple STELLA model of the motion of a projectile near the surface of the earth. The projectile has an initial velocity with positive x and y components, and a constant downward acceleration. The boxes represent the numerically integrated quantities: the components of the position and the y-component of the velocity. The position components are linked to flow icons representing velocity; the y-velocity box is linked to its rate of change, the acceleration. A graph icon produces the graph in the lower window, which plots the object's path in x-y coordinates when the model is run. Users assign initial values to the integrated variables and functional relationships to the rates of change; also they can define any other needed parameters. As the quantities are defined and linked together, the software automatically generates equation-like code indicating the relationships, as would be seen in a programming language. Figure 2.4 shows the code generated by the projectile motion model.
Figure 2.3. A STELLA model of projectile motion. The upper window contains the icon-based model. The lower window shows the trajectory in x-y coordinates. Created with STELLA 8, High Performance Systems, Inc. See http://www.hps-inc.com
The Bremen researchers have used STELLA in upper level high school physics courses, where groups of students would work with the software on a common problem, such as kinematics with air resistance, or Rutherford scattering. Schecker has discussed the types of classroom and curriculum transformations that were made resulting from the use of STELLA. Content shifted from solving close-ended, "canned" problems to open-ended investigations by students, and the amount of interaction between students and between students and the instructor increased (Schecker, 1994). In the classroom setting, students began to see the computer and the software as a tool for exploring physics, rather than a novelty, in just a few exercises. (Schecker, 1993a).

More importantly, Schecker states evidence that the use of STELLA affected students' conceptual understanding of physics, or as he calls it, their "physical competence," as well as their approaches to new problems. Part of this physical competence that the STELLA-based course emphasized was a way of qualitatively assessing a physical situation, as opposed to immediately plugging numbers into equations. (This is a more expert-like strategy; see section 2.2.) Another key idea was the problem solving strategy of (1) finding the interacting bodies in a system, (2) quantify
the interactions with forces, (3) calculating the resulting force and acceleration on the body, and (4) calculating the "kinematic chain"—that is, updating the velocities and positions. In other words, this is essentially the iterative application of the momentum principle that I outlined in section 1.3. Schecker asks whether students will be able to apply this strategy from computer modeling "not only when a computer is available but also if they reflect on a new problem posed in an experiment" (Schecker 1993a).

He and his colleagues interviewed students from two "advanced level" physics courses that had made use of STELLA. Pairs of students were presented with a problem that they had not seen in class: a cart rolling on a nearly frictionless surface is accelerated from rest by a soft spring, travels at a nearly constant velocity, then rebounds off of a wall by a hard spring attached to it, reversing direction. The student pair was given ten minutes to repeat the experiment and sketch a velocity-time graph. No computer was available during the interview for students to use. Student discussion and their presentation of their solution was audio recorded. Based on transcripts of student responses, the researchers classified three different "levels of competence:"

1. those who applied the above problem-solving strategy on their own;
2. those who try to, but need help, and;
3. those who resorted to invalid formulas (such as $s = at^2 / 2$), or recall the graph from memory.

Schecker states that about half of the 20 students interviewed demonstrated competence level 1. However, the example he uses as evidence does not make it clear what types of student responses explicitly show this competence level. Still, this it at least sets a precedent that students can make use of computer modeling on their own when faced
with new problems. It is also interesting that some students made use of formulas, such as constant force motion, that did not apply to the current situation. Later, I will show similar results from my own studies.

2.1.3 Boxer

Boxer is a computational environment, complete with its own programming language, that was designed by researchers at the University of California at Berkeley. It was an outgrowth of research and development of an earlier computer language called Logo (diSessa, Abelson, & Ploger, 1991). Logo, itself a dialect of LISP, was developed by Seymour Papert at MIT in the late 1960's, (Papert, 1980) in cooperation with the Bolt, Beranek, and Newman company, under the direction of Wallace Feurzig. In the 1980's, Logo was widely used in elementary and middle schools as a way to introduce students to computer programming and graphics, as well as geometric concepts. It was not as widely used at the university level, although there were a few cases where university physics instructors used it for modeling and visualization (see for example Lough, 1986). Boxer was designed to overcome some of Logo's drawbacks and extend its capabilities; for example, while young children could learn to write Logo programs that created geometric shapes, the language was too difficult to learn to do much else. Also, Boxer, unlike Logo, was designed to be more than a programming language; it was intended to be a flexible, multi-purpose, graphical computational environment (diSessa, Abelson, & Ploger, 1991).

Boxer, like STELLA, is a graphical medium, but it is much more generalizable and powerful than STELLA. Boxer users write programming code line by line, but that code is organized and linked together in graphical ways. For example, procedures are
contained in "Doit" boxes. A Doit box can be given a name, which allows it to be
accessed, like a subroutine, by another procedure. Variable and parameter declarations
are stored in "Data" boxes, and graphical displays that make use of data and procedures
are contained in "Sprite" boxes. An example of a complete Boxer program that models
projectile motion near the earth's surface is shown in Figure 2.5.

The Berkeley Boxer Group has written extensively on the educational effects of
the use of Boxer among students ranging from elementary school to college level. Some
of this work dealt with students programming in Boxer to explore concepts in physics. In
one case, the group taught a high school physics course where Boxer was used in
classroom collaborative design tasks (Sherin, diSessa, and Hammer, 1993). Students in
this course were taught the basics of programming in Boxer during the first five weeks of
the course. The next ten weeks focused on exploring five core physical situations, such
as dropping an object or striking a puck on a frictionless surface. Students in this course
performed a wide variety of activities to develop understanding of these situations,
including programming models in Boxer, using "microworld" simulation software
designed by the instructors, and drawing graphs of motion. During classroom discussion,
Boxer also was used a medium for exploring ideas about motion. Students would discuss
ideas about how objects moved; the instructor provided help, when needed, in translating
students' ideas to the precise language of the computer. The instructor would then
program the proposed ideas into the model, and the resulting visualization would further
discussion as to whether the computer model was suitable. The goal was, in effect, to get
the students to "design Newton's laws" by continually revising the computer model
(diSessa, 1995).
Figure 2.5. Boxer program that models the motion of a projectile near the surface of the earth (constant downward acceleration, no air resistance). Created with PyxiBoxer by Andrea A. diSessa and Edward H. Lay, PyxiSystems LLC. See http://www.pyxisystems.com
In another study, which was closer to M&I's instructional model, Sherin examined in detail the implications of replacing algebra as the main symbolic representation in physics instruction with programming, specifically, programming in Boxer (Sherin, 2001). For his experiment, Sherin had 10 pairs of volunteers from the 3rd semester introductory physics course for engineers at UC Berkeley. These students had already completed the first two semesters of the course, which covered mechanics and electromagnetism, respectively (Sherin 1996). Five pairs of students were placed in an algebra group, where each pair solved physics problems for about 5.5 hours. Most of these problems were typical introductory kinematics or mechanics problems, and one dealt with electric circuits. Some questions required students to interpret algebraic expressions or to think of physical situations that are described by certain equations. The other five pairs were placed in a programming group. Each pair in this group was given about 4 hours of training in programming in Boxer. After this each pair worked together for 4 additional hours on computer problems that required programming simulations of physical systems such as a tossed ball (both ignoring and including air resistance), a mass on a spring, and a block pushed on a table. Although most of the students in the programming group had some prior experience in programming, the amount of experience varied widely; some had only brief instruction in high school, some had taken a college programming course, and one was a computer science major who could program in several languages (Sherin 1996).

Sherin's initial conjecture is that programming might be easier for students to understand than algebra, because programs can be "mentally run." In the end, his conclusion on this point is equivocal; students were able to interpret algebraic equations
as well as programs, though a program, with its multiple lines and subsections, can give context that helps students interpret parts of it. His second conjecture is that programming might favor a different "intuitive vocabulary." That is, replacing algebra with programming may not just be a case of exchanging one tool for another that leads to the same type of conceptual understanding of the subject. Instead, programming may change the very nature of the conceptual understanding. In his observations, he finds that physics using algebra supports a vocabulary of "balance and equilibrium," while programming supports a vocabulary of "process and causation."

To understand this, it is necessary to understand Sherin's theoretical framework. This framework is based on two conceptual constructs that he developed in his dissertation (Sherin, 1996). The first, called "interpretive devices," are natural language utterances that interpret symbolic statements. To take an example from algebra, a student derived an equation $x = mg/k$ to describe the relationship between an object of mass $m$, hanging motionless at the end of a spring of stiffness $k$, and the distance $x$ that the spring stretches, where $g$ is a constant. The student described this equation as making sense by saying

*As you have a more massive block hanging from the spring, then your position $x$ is gonna increase, which is what this [expression] is showing. And that if you have a stiffer spring, then your position $x$ is gonna decrease. That's why it's in the denominator (Sherin, 2001).*

In other words, the student mentally varies a single parameter, while holding the others fixed, to imagine the effect on the result. This method of making sense of the expression is a type of "narrative" interpretive device that Sherin calls "CHANGING PARAMETERS."

The second construction he calls "symbolic forms." These are specific features and patterns in symbolic representations that people pay attention to. For example, an
algebraic pattern of the form __ = __ is a symbolic form called "BALANCING", since it triggers students to think about two balancing or counteracting forces, influences, or terms. In coding and classifying students' verbal utterances when working the problems, Sherin developed a catalog of interpretive devices and symbolic forms, and analyzed how they differed in algebraic and programming representations. These differences are of particular interest to the present research.

First of all, Sherin notices that two interpretive devices appear almost entirely in programming alone. In fact, these devices constitute a large part of all the interpretations of programming that students made. The first device, called "TRACING", which appears solely in programming, refers to instances where students verbally step through a program one line at a time; that is, their narration of what happens in the program "follows the timeline structure of the written program" (Sherin, 2001). An example is when a student reads a line in the program, describes and points to the change that line makes in the graphical display, such as changing the position of a dot, and continues to do this in succeeding lines. The second device, called "PHYSICAL CHANGE," refers to when students picked one or two lines in a program containing expressions and followed how the quantities in the expressions vary through time. "PHYSICAL CHANGE" differs from "CHANGING PARAMETERS," mentioned above. In "CHANGING PARAMETERS," students looked at an expression and imagined how quantities would differ in different circumstances, e.g. when an object with different mass is hung from the spring in the example above. In "PHYSICAL CHANGE," students think about how the quantities that describe a single situation change in time, e.g. how the acceleration of an
object or the force of air resistance on an object changes as the object moves through the
air.

Sherin also compares symbolic forms in programming and in algebra. The most
important finding is that an instance of "BALANCING," described above, never occurs
in programming. This is most likely due to the different capabilities of these two
languages. In algebra, it is easy to write an equation that indicates the symmetric
balancing, such as $F_1 = F_2$ indicating two balanced forces. But it is difficult to do so in
programming because programming tends to focus more on "assignment" than "equality"
(these are not terms that Sherin uses, but he alludes to the concept). Writing the
statement "$F_1 = F_2$" in VPython, for example, does not indicate a balanced equation;
rather it is an instruction that tells the computer to assign the name $F_2$ to the object
currently named $F_1$. (The technical details of assignment in Python differ somewhat
from other languages and will not be discussed here.) Knowing the difference between
assignment and equality is even more important when trying to understand a VPython
statement such as "$x = x + 1$", a line that looks algebraically impossible, but really
means "assign to $x$ the value of its old value plus one." This difference may be more
clear to users of Boxer because the syntax is quite different; what is written as "$x = x + 1$"
in VPython would be written as "$\text{\texttt{\textbackslash \texttt{change} } x x + 1}$" in Boxer. As Sherin mentions, the only
place where balance occurs in programming is in conditional statements, such as a line
that says "$\text{\texttt{if} } F_1 = F_2 \text{\texttt{, then execute some instruction.}}$ (Note that some languages, like
Python, use "$==$ for testing equality; other languages use "$\text{\texttt{:\texttt{=}}}$ for assignment.)

Sherin distinguishes some symbolic forms that are specific to programming. Of
particular interest to this dissertation is one he calls "SETUP-LOOP." He claims that
students can construct procedures in programs by remembering the general ordered structure of programs; that is, lines in a program are executed in sequence. If there is, for example, two sequential stages of motion in the motion of an object, the processes that model these stages would occur in the same sequence, one after the other. This general form of "SEQUENTIAL PROCESSES" has a special case of "SETUP-LOOP," which is the general structure that models motion. In every motion-modeling procedure, the first step is that the variables that describe the motion are set up; for example, the initial conditions of position and velocity are declared. Following that is a list of statements in a loop that change the relevant variables repetitively; for example, the iterative application of the momentum principle. In M&I, getting students to understand this "SETUP-LOOP" structure is central to their success in computer modeling. Later in this dissertation I will discuss student difficulties with this structure, and the instructional means that were designed to help alleviate this difficulty.

2.2 Expertise in solving physics problems

Research into the differences between experts and novices in solving physics problems is germane to my investigations into the differences in problem-solving approaches between Matter & Interactions students and traditional course students. Evidence from cognitive research shows that experts in physics differ from novices both in their problem-solving strategies and in the way they classify or categorize physics problems. Larkin and colleagues did a great deal of research on the cognitive aspects of solving problems in physics in the late 1970's and early 1980's.
As Larkin (1981) points out, students in physics are taught various equations and principles, but rarely do textbooks point out when, or in what situation, it is useful to apply specific principles. As a result, even good students who have learned the presented principles may have no way of deciding what principle to use when faced with a new problem. Novices then tend to rely on "weak" problem solving methods, ones that are not specific to the domain of physics, but are "generally useful for searching a large set of objects (here principles)" (Larkin 1981). One of these methods is called means-ends analysis. In means-ends analysis, the problem-solver attempts to eliminate the difference between the current state and the goal state. The solver does this by choosing relevant steps or techniques, called "operators," that can minimize the distance to the goal. If, in trying to apply an operator, the solver still finds a difference to the goal, or a missing function that is required to successfully use the operator, the solver then creates a sub-goal of eliminating this new difference; essentially, the new goal or "end" becomes solving a smaller problem within the larger one. Thus, means-ends analysis is a cycle of setting subgoals, often nested subgoals within subgoals when necessary, whenever a means to reaching the main goal is blocked. (For more discussion, see Anderson, 2000.)

Here is a simple example of a means-ends analysis. Consider a kinematics problem that gives numerical values for the initial velocity components $v_{0x}$ and $v_{0y}$ of a projectile launched from the surface of the earth. The goal state might be to find the horizontal distance $x$ from the starting point to where the projectile lands on the ground. The solver decides to use the equation $x = x_0 + v_{0x}t$, since solving this equation will reach the goal. But the operation is blocked because the time of flight $t$ is not known. So the new subgoal is to find $t$, and to do this, the solver chooses the equation $v_y = v_{0y} - gt$. This
operation is blocked in two ways: $v_y$ is not known, and $t$ is not alone on the left side of the equation. So another subgoal is set: find $v_y$. Perhaps at this point the solver uses a remembered fact: $v_y = -v_0y$ when the projectile reaches the ground. Then, the next subgoal is met by rearranging the equation to read $t = 2v_0y/g$. After plugging in the given numbers into this equation, the first subgoal of finding $t$ is met, so the solver goes back to the original equation, and solves for the main goal of $x$.

The particular case outlined above is also an example of working backwards. Larkin, McDermott, Simon, and Simon (1980) found that one key difference between expert and novice problem-solvers in physics is that experts work forwards while novices work backwards. When working forwards, experts start with the given quantities in the problem, and then apply equations that make use of these given quantities until the desired quantity is reached. Novices, on the other hand, start with equations and principles that, if solved, would give the desired quantities. If these equations cannot be immediately used because they contain unknown parameters, they will then switch to means-ends analysis, until the given information is used. However, Larkin et al. also mention that experts work forward only on problems that are easy for them; that is, on problems on which they have enough experience to know that such a strategy will work.

This does not necessarily mean that physicists solve problems that are new to them using the same type of means-ends analysis that a novice would use, because experts and novices may organize their knowledge in different ways. In an experiment that involved observing a physics professor and a good undergraduate student solve mechanics problems, Larkin and Reif (1979) constructed models to characterize and predict the problem solving processes of experts and novices. They found that the first
step in both the novice and expert models is to construct an "original description" of the problem consisting of a labeled diagram and a list of known and unknown quantities. But after this, the models vary drastically. The novice's next step is to immediately apply mathematical equations in a means-ends way. The expert, on the other hand, selects a general method and uses it first to construct a low-level, qualitative description of the problem, while checking to see if the method will cause any difficulties. Once this is done, the expert then begins to work out in detail the mathematical solution.

Furthermore, while the novice treats all equations and principles at the same level of importance, the expert's general methods are organized around central principles, such as Newton's second law. Linked to these central principles are a larger number of subsidiary principles, like force laws and kinematics equations.

Further evidence that experts organize their knowledge around fundamental principles comes from Chi, Feltovich, and Glaser (1981). Through a series of studies they attempted to learn how problem-solvers categorize problems. Approximately 20 introductory mechanics problems were given to eight PhD students in physics and eight first year undergraduates, who had just completed a mechanics course. The subjects were asked to sort the problems into categories based on "similarity of solution" and to explain their choice of categories. The novice physics students tended to sort the problems into categories based on "surface features," such as the objects referred to in the problem statement (e.g. springs, pulleys, inclined planes), the relationships among these objects (e.g. a block on an inclined plane), or the physics terms mentioned in the problem (e.g. friction, center of mass). The categories created by the experts, on the other hand, were
organized according to the major physics principle that was necessary to solve the problem, such as Newton's Second Law, or conservation of energy.

Recognizing that these categories may indicate the structures in which the expert and novice knowledge is organized, Chi, Feltovich, and Glaser ran two other studies. In the first, they asked both experts and novices to state all they could about how to solve problems of various types; the names of these types were taken from the categories created by the experts and novices in the previous studies. Experts tended to state rules and conditions for solution procedures based on fundamental physics principles, and these rules were used early in the solution description. Novices' rules were closer to "attempts to find specific unknowns, such as 'find mass'" (Chi, Feltovich, and Glaser, 1981). In the second study the experimenters asked the experts and novices to describe their "basic approach" for each problem. Experts could sum up their basic approach in a principle such as $F=ma$, conservation of energy, or Newton's third law. Novices' basic approaches could not be classified in this way; they instead could only give vague, general, descriptions of how to proceed (e.g. "figure out what was happening," "think of formulas") or list in detail the set of equations they would use.

The evidence from these studies shows that experts organize their knowledge of physics around fundamental physical principles. The structure of this knowledge is hierarchical; a small set of principles is recognized as primary, and these cue subsidiary principles and equations. But novice physics students, even skilled ones who have completed an introductory physics course, do not gain this type of knowledge organization. At first glance it may be unfair to expect physics instruction to make students' knowledge structures expert-like, considering that experts have had years of
practice. Eylon and Reif (1984), however, have shown that explicitly teaching material organized in a hierarchical way can help novice students. In one experiment, subjects had to learn the details of how to derive the gravitational acceleration $g$ from measuring the motion of a bouncing ball. The argument was presented to one group in a hierarchical way; that is, an overview of the major steps was first given, followed by the detailed steps that fill in the overview. Another group was presented the argument with all steps at the same level, and a third was given this same non-hierarchical treatment twice. The hierarchical treatment group performed consistently better than the one-pass low-level group in complex problem-solving tasks related to the material they studied, and either better or as well as the two-pass group.

The *Matter & Interactions* curriculum attempts to make use of these results in its treatment of problem solving. Instructors emphasize that there are only three fundamental mechanics principles: the momentum principle, the energy principle, and the angular momentum principle. Once one of these principles is chosen for analyzing a system, it should cue a lower level of subsidiary principles that can be used to "flesh out" the unknown terms of the problem. For example, once the momentum principle is chosen, it can cue the use of gravitational or electric force laws, the definition of momentum, and the definition of velocity. The energy principle could cue the use of the definition of kinetic energy, the definition of work, or various potential energy functions for different systems. Armed with this type of hierarchy, even if students do not have a clear idea of how to solve a problem, they can always pick one of the three principles to start, and switch to another if it does not give useful information. The "search space," or set of legal steps to solve a problem, is greatly reduced, as opposed to the situation for
students from a traditional course, who might consider all principles and equations at the same level of importance. This, along with the centrality of the momentum principle in computer modeling, motivates investigating whether students in M&I can approach problems using fundamental principles and equations organized in this hierarchical way, and if they differ from students in traditional courses (see research question 4 in section 1.4).

2.3 Research methods involving verbal data

The studies that will be discussed in this thesis relied on qualitative data, including students' written solutions to problems, students' VPython code, and verbal data. Participants in the studies spoke, sometimes without prompting and sometimes in response to the interviewer, while working on assigned tasks. Educational researchers have often used verbal data from interviews and verbal protocol studies as a way to shed light on subjects' thinking processes and problems solving abilities. A number of research techniques can be used to enable subjects to verbalize or report their thoughts, and each has its strengths and weaknesses in terms of the information gained by the researcher, and the validity of that information. Below I will describe some of the techniques that are relevant to this research.

2.3.1 "Think-aloud" protocol analysis

Ericsson and Simon (1993) have examined in great detail the various methods of verbal reports and analysis. They define "think-aloud" protocols to be a type of "concurrent verbal report"; that is, the subject talks out loud while performing a task or solving a problem, thereby verbalizing the information that is attended to, or "heeded," at
the time it enters short term memory. Think-aloud protocols are regarded as the most valid way of accurately reporting what is happening in the subject's short term memory during the problem-solving process; the utterance of such verbalizations do not appreciably affect or interfere with the cognitive processes that occurring. Retrospective reports are considered less valid than concurrent reports. In the ideal case of a retrospective report, a subject verbalizes everything he or she remembers about solving a problem immediately after the problem is finished. If the necessary retrieval cues remain, a subject can give an accurate verbalization of memory structures, but the retrieval process is not perfect, and subjects may bring in additional information from long term memory. When subjects can't remember a certain piece of information, they may tend to fill in what they can't remember with what they believe they "must have thought." In fact, people are notoriously poor at introspectively describing their own cognitive processes. In a thorough review of psychological literature, and in their own experiments, Nisbett and Wilson (1977) have shown that subjects are often unaware of the stimulus that directly causes them to respond in a certain way on various tasks. When asked to report on why they responded the way they did, subjects will give instead causal theories invented "on the spot," based on what they think is most plausible.

Also, outside influences can interfere with subjects' naturally expressed verbalizations, thereby threatening validity. For example, the instructions that the experimenter gives, depending on the wording, could encourage subjects to do extra processing of information before verbalizing it. If subjects believe that the experimenter expects them to make a certain response, this too could cause them to filter their thoughts before verbalizing. At times, subjects may stop verbalizing; in such cases it is best for
the experimenter to gently remind the subject to keep talking, and not to give directives that may unduly influence short term memory.

In the two experiments that were conducted, students were instructed to give concurrent "think-aloud" reports; that is, they were ask to talk out loud while working on the tasks, saying anything that came to mind. Experiment 2, however, deviated from a traditional think-aloud protocol study. Due to the complex nature of the task, more interaction occurred between the students and the interviewer. (See section 4.4 for more details.) This does not necessarily invalidate the results. To gain understanding into student cognition, many educational studies have relied on analyzing dialogues from interview sessions that were more interactive than think-aloud protocols. These include demonstration interviews and tutoring sessions, described below; the dialogue in experiment 2 had the character of both of these types of research methods.

2.3.2 Demonstration interviews

Demonstration interviews are a common technique in physics education research for learning about student difficulties or misconceptions. In these sessions, students perform an experiment or do some task, or observe an interviewer or instructor perform a task. The interviewer then asks students to explain the phenomenon or situation, paying attention to where student explanation of the phenomenon deviates from standard scientific explanation. Driver (1973) was one of the first to use this technique in the context of physics education. She observed a teacher working with four students in a laboratory classroom in which the students learned about the concept of force. Students conducted experiments, watched demonstrations by the teacher, and discussed results with each other. Through the teacher's questioning and the students' reactions, it was
found that students often believe inanimate, non-moving objects cannot exert forces (for example, students believed a chair does not exerting an upward force on a person standing on it.) Minstrell (1982) examined similar student difficulties with similar research techniques. He performed similar demonstrations and asked probing questions of students in his own high school physics classes, which he tape recorded and transcribed.

The Physics Education Research Group at the University of Washington has also made extensive use of demonstration interviews. For example, Trowbridge and McDermott (1980) used demonstration interviews to examine student understanding of kinematics. During the interviews, students observed various motions of real objects and were then asked a series of structured questions about their observations. The interviews were then transcribed and analyzed. Student difficulties in many different topics in physics have been explored in this way by the UW group, including work-energy and impulse-momentum theorems (Lawson and McDermott, 1987) and electric circuits (McDermott and Shaffer, 1992).

2.3.3 Tutoring sessions

As I will describe in chapter 4, the interview sessions in experiment 2 often had the character of tutoring sessions. The quantitative analysis of the dialogue from these sessions involved categorizing and counting statements and prompts by the interviewer. This technique has also been used in educational research literature. For example, Lepper and colleagues have performed a series of studies focused on tutors' interventions. They have coded and analyzed tutors' verbalizations, actions, and decisions, looking both at the information tutors give to students and the motivational strategies tutors use.
Putnam (1987) analyzed teacher-student dialogue from tutoring sessions on arithmetic problems to examine how teachers adjusted to student responses, answers, and errors. His analysis followed a pattern that was similar to protocol analysis methods used to study problem solving. Chi (2001) analyzed both student and tutor verbalizations from tutoring sessions to examine whether the effectiveness of tutors comes from tutors' effective diagnoses, students' construction of knowledge based on tutors' inputs, or the interaction between student and tutor. While the goal of experiment 2 was to examine student difficulties, and not the effectiveness of tutoring in correcting these difficulties, these previous studies show that tutor's verbalizations are a valuable source of qualitative data, and that there is a precedent for the analysis of dialogue from tutors interacting with students.

2.3.4 Verbal analysis

Chi (1997) describes a technique of analyzing verbal reports that is different from traditional protocol analysis both in its goals and methodology. In traditional protocol analysis, the focus of the research is to capture the processes of problem solving or decision making. Typically, the task is analyzed in a detailed, fine-grained way, so as to have a complete description of all possible solution steps that could be taken (often called the "problem space"). Once the problem space is mapped out, a runnable computational model is created that simulates the path that is taken through the problem space to reach a solution. As Chi states:

Then, the goal of protocol analysis is to see whether there is a match between the path that a solver took and the sequence of states that a simulation model generates. If not, then one can tweak the model so that it simulates an actual solver's path. (Chi, 1997)

Chi's form of verbal analysis, however, is less confirmatory and more exploratory. It
differs from protocol analysis in that:

…the protocol analysis method starts with a model of the task, which can be referred to as the ideal template. The goal of the method here, in contrast, is to seek the model that a subject has, without creating an ideal template a priori.

The theoretical perspective behind verbal analysis is not centered on representing "ideal" knowledge that can be interpreted by a computer, but rather in uncovering what a learner knows and how that knowledge influences the learner's choices, whether they are correct or incorrect. This knowledge is uncovered through careful analysis of the learner's verbalizations, paying careful attention to their overall structure, that is, categorizing utterances into propositions, concepts, goals, or rules. The problems that students in my study solved are complex and novel enough that it makes little sense to try to construct a computational model of how to solve them beforehand. My interest is in observing and uncovering the difficulties that students have with these challenging problems, categorizing these differences, and seeing whether or not patterns emerge in the solutions of students of different instructional treatments. For these reasons, my theoretical perspective is similar to that of the verbal analysis method. However, in order to maximize validity and reduce biasing the subjects' thinking process, students in my experiments were instructed to give concurrent think-aloud verbalizations, as described by Ericsson and Simon. Chi's verbal analysis can be used on these types of utterances, as well as other verbalizations that would be considered less valid in the protocol analysis perspective, such as self-explanations.

Chi advocates a method of analysis that combines both qualitative and quantitative methods so as to minimize the shortcomings of both. Qualitative research methods can provide a richer, deeper understanding of the area of interest, but the drawbacks can be a lack of objectivity and replicability. Quantitative methods are more
objective, but they can only make conclusions about the narrowly defined hypothesis at hand; thus, results from sterile laboratory experiments may lack generalizability to real world situations. Chi mentions several ways to combine the two methods, but the verbal analysis method that is described in detail is one where:

qualitative data is examined for impressions and trends, methods of coding are developed to capture those impressions, and the codings can then be analyzed quantitatively. (Chi, 1997)

The specific steps of the analysis consist of the following:

1. **Reducing or sampling the protocols.** The sheer amount of verbal data can be overwhelming, so choices must be made to reduce the amount to be coded, either by random sampling, or through preliminary coding on the whole amount, followed by a more detailed coding on a subset.

2. **Segmenting the reduced or sampled protocols (sometimes optional).** Here, the protocols are divided up into convenient units of analysis.

3. **Developing or choosing a coding scheme or formalism.** The verbal data is then examined and coded based on the theoretical perspective of the researcher, the hypothesis that is being tested, and the type of task being examined.

4. **Operationalizing evidence in the coded protocols that constitutes a mapping to some chosen formalism.** That is, the researcher must then decide what coded utterances constitute evidence of the phenomenon in question based on the theoretical perspective.

5. **Depicting the mapped formalism (optional).** This can help with communicating the data as well as detecting patterns in the data.

6. **Seeking patterns in the mapped formalism.** Ways to quantify the patterns
should be used here.

7. *Interpreting the patterns.* This, again, depends on the researcher's theoretical orientation and hypotheses.

8. *Repeating the process, perhaps coding at a different grain size (optional).*

This basic method is used as a guideline for my own analysis, particularly in experiment 2 (chapter 4). I will present the specific analysis methods that I used later in this dissertation.
3 Experiment 1: The "hard problems" study

In 2001 I conducted a study to address the question of whether students in the M&I course see computer programming or iterative methods as a valid problem-solving tool and use them when faced with new, challenging problems (question 1 from section 1.4). Preliminary results showed qualitative differences in the way students from a traditional introductory physics course and students from the Matter & Interactions course approached the complex problems used in the study. Thus, a secondary research emerged interest of how the two classes differed with respect to their use of fundamental physics principles (question 2 from section 1.4). In this chapter, I will first discuss a small pilot study that preceded the main experiment. I will then discuss the participants, procedure, analysis, and results of the main experiment.

3.1 Pilot study

A pilot study was conducted in the Summer of 2001 at Carnegie Mellon University that served several purposes. It was an initial attempt to test the hypothesis that students from the M&I course would invoke an iterative application of the momentum principle when solving dynamics problems, and that students from a traditional physics course would not. (It may seem obvious that students from a traditional course would not use iterative methods, since these techniques are not taught, but observing can at least serve as a "reality check" that computer modeling or iteration is too complicated a strategy for introductory students to construct on their own, without explicit instruction.) The pilot study was also conducted to test for any problems in the
experimental design. Several elements of the experimental design were altered for the main study as a result of lessons learned in the pilot.

3.1.1 Pilot study: participants

Four students who had completed the first semester of a traditional introductory physics course at Carnegie Mellon University volunteered to participate. Three students who had completed the Matter & Interactions-based introductory mechanics course also volunteered, as well as one physics graduate student who had never taught or been exposed to the Matter & Interactions curriculum.

3.1.2 Pilot study: procedure

Each participant met with me individually and privately for an experimental session. During the session, participants were asked to outline solutions to physics problems that involve predicting the position of an object at a future time. Five problems were given; they are listed in section 7.1 of the Appendix. Problem 1, a projectile motion problem, and problem 3, involving a satellite in a circular orbit, were computationally easy enough to solve using analytic methods learned in a traditional physics course. Problems 2 and 4 were modified versions of problems 1 and 3, respectively. Problem 2 modified the projectile motion problem to include air resistance, while problem 4 altered the orbit problem to include a brief force on the satellite due to a rocket engine. These, along with problem 5 (a block on a spring moving in two dimensions), could only be solved through an iterative approach--either on a computer, or by following the computer modeling procedure by hand. The students were not asked to actually solve the problems (although they sometimes did). Instead, the students were told explicitly that they might
not be able to solve some of the problems; therefore they should write the steps they
would take, as if they were "trying to get partial credit on an exam." Students were told
to imagine that any technology was available, such as a computer or a graphing
calculator, and that they could explain how they would use such devices in their
approach, if they wished. No computer was actually provided. Students were also
provided the textbook from their respective course and told that they could consult it at
any time. Participants were instructed to "think aloud" while working, saying anything
that came to mind. Immediately after the students worked as far as possible on each of
the problems, they were asked follow-up questions based on the work they produced or
the approach they chose. Specifically, they were asked if technology would have helped
in their solution, and, since some students indicated integration would be necessary, they
were questioned further about the role of integration.

3.1.3 Results of pilot

None of the traditional course students made use of or suggested using an iterative
application of the momentum principle, supporting the original hypothesis. In post-
session debriefing, when asked if they believed a computer could help solve the
problems, two students thought that a computer algebra package like Maple might be able
to help solve the problems, while the other two did not believe a computer could help.
None of the students had ever used either a computer language or a software package to
solve a physics problem. As expected, these students were not able to invent or construct
an iterative technique; it never occurred to them. It is useful to rule this possibility out,
since an expert of sufficient skill can construct such methods.

For example, the physics graduate student, after an hour of thought and work, was
able to construct an iterative process. For problem 2, without any prompting or
questioning from me, he suggested using an iterative approach, saying:

Would you have to do it iteratively, maybe? Would it be the easiest way? At each step the ball
takes, there are new velocities, which means a new air resistance on it, which means a different
problem at each little step. So yeah, I'd do it iteratively. Let's see if I can set it up.

He struggled to come up with the actual steps, however. He tried using a single equation
to predict the positions, which ended up looking like constant acceleration projectile
motion equations. He was not satisfied with his solution:

I want to do it iteratively, take a small step, calculate my new x and y, new v_x and v_y, put them
back in to get the next x and y, but I just can't see...the general process.

Unable at this point to set up the equations, he stated what he would do if he had them:

Once I got the equations set up, I would use Excel or something and put the equations in there and
run through, do it iteratively, read off whatever time, read off all the values.

He continued to struggle with this idea throughout all of the problems. Finally at the end
of problem 5, I asked him to go back to this process. At this point, without further hints
from me, he was able to reconstruct the steps of the iterative process: use \( v = v_0 + (F/m)\Delta t \)
to update the velocity components, then use \( \Delta x = v \Delta t \) to update the position components.

Like the graduate student, and in contrast to the students from the traditional
course, the students who had taken M&I were able to make use of iterative methods on
the harder problems. However, it was not often their first choice of solution. It was
usually only after prompting from me that they decided to write the necessary steps in
"pseudocode" that would model the behavior of the system. At times they expressed a
preference for analytical methods, and a feeling that using computation to solve the
problems was inferior.

One of the M&I students ("Doug") decided that he would use a computer to solve
the complex problems, without any prompting from me. For problem 2, he said:
I would definitely use a computer for this, and I would do one of those things where you have a little delta t, and you step it over and over while you're changing velocity according to these formulas...

He then wrote out the equations used in the iterative procedure. For problem 3, he originally wanted to write a program, but then realized the problem could be solved analytically. For problem 4, he discussed whether the additional impulse from the rocket would make a noticeable difference in the satellite's orbit, and decided one approach would be to write a program, and see if the motion is noticeably different from an analytic approximation. For the last problem, he again decided that writing a program would be the best choice, but he did at first state that he felt it would be a "cop-out to write the program."

Another M&I student ("Kurt") relied on invalid constant acceleration equations in problem 2 initially. He used the equation for constant acceleration motion (that is, $at^2/2$), but he used the air resistance force to find the acceleration. The air resistance force, and therefore the acceleration, is not constant, because it depends on the velocity, which is changing. When I explicitly pointed out to him that the force expressions depended on the velocity, he then realized they were not constant, and began to think about using a computational approach. I then asked if he could write out the steps needed for a computer model, and he was able to do so. He did not use an iterative approach to problem 4; instead he tried using a combination of energy equations and constant acceleration equations, but got stuck. For problem 5, he started out by saying that he did not want to "resort back" to using computational methods, but when I then reminded him that he was allowed to imagine using any technology, he decided to write a pseudocode program.
The third M&I student ("Gordon") tried to write analytic integrals for problem 2. While he had the essentials of integrating the force over time to get a function momentum, and then integrating velocity (momentum divided by mass) over time to get a position, he realized that the problem was too complicated to solve. After being asked if a calculator or computer could help, he realized he could write a program, and verbally described the steps involved. In problem 3, he decided that a computer model could be used again, and when I asked him to outline the steps of the program, ignoring syntax issues, he was able to write a pseudocode program. He then used this approach in subsequent problems.

The pilot was useful in pointing out ways the experiment could be refined. Two issues emerged with the design of the pilot. First, the fact that two of the hard problems that required the iterative procedure were preceded by similar easy problems that could be solved analytically may have biased student performance. By solving these easy problems first, students may have fallen into a "mental set"; that is, they got used to the idea that analytical methods would work with the given problems, and therefore may have been prevented from thinking of alternative methods. For example, many students used the formulas for constant acceleration projectile motion on problems where they were not valid; this may have been because they used them successfully on the first problem. To deal with this, in the main experiment, only the "hard" problems, the ones that required iterative methods, were given to students. Various changes were made to the order and minor surface features of the remaining problems as well. For example, the orbit problem now came first, and the projectile motion (with air resistance problem) was
given second, so as not to bias students into using constant acceleration formulas. (As we will see later, students still often used these invalid formulas).

The experimental setting itself may have contributed to this same "mental set" effect. Even though students were told to imagine that they could use any technology, no computer was provided for their use. Only whiteboards and markers were present, which perhaps suggested to the students that analytic methods would be enough. This is somewhat artificial; in the *Matter & Interactions* class, problems that required computer modeling were always assigned in a computer lab, where students could readily write VPython programs. This issue was dealt with in the main study by providing a computer for students to use in the experimental setting. The instruction to "outline" a solution was also artificial. In the main study, students were simply told to try to solve the problems using any means necessary.

There were also some issues with the placement and timing of interviewer questions. The students were asked after the three "hard" problems whether technology could help in their solution. In the case of the *M&I* students, this prompted them to think about writing a program. It is not clear that Kurt or Gordon would have thought of using a computer model to solve these problems had I not given these hints. Furthermore, once the hint was given, it likely influenced their solution method choices in subsequent problems. In the main experiment, I took a more "hands-off" approach: students were not questioned about their choices until the end of the session, after all the problems had been attempted.

Following the pilot study, the main phase of the hard problems study was performed in the Fall of 2001. I will now describe the details of the main experiment.
3.2 Participants

Participants for the study were paid volunteers recruited from two different calculus-based, first-semester introductory physics courses at Carnegie Mellon University in the Fall 2001 semester. The first was a course for natural science and computer science majors, consisting of approximately 70 students, that made use of the *Matter & Interactions* text (Volume 1, Modern Mechanics) and curriculum. The second course was a "traditional" physics course for engineering students, which made use of a traditional textbook (Young & Freedman, 2000), and consisted of about 190 students. Six students from the traditional course and five from the *M&I* course participated in the study. The study took place at a point in the semester when all of the participants had completed the classical mechanics portion of their respective courses (in the case of the *M&I* students, up to but not including angular momentum). Tables 3.1 and 3.2 list the students by pseudonym, gender, final grade in the respective course, and major.

**Table 3.1.** Participants from Traditional course.

<table>
<thead>
<tr>
<th>Pseudonym</th>
<th>Gender</th>
<th>Grade</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sam</td>
<td>M</td>
<td>A</td>
<td>Eng.</td>
</tr>
<tr>
<td>Jerome</td>
<td>M</td>
<td>B</td>
<td>Eng.</td>
</tr>
<tr>
<td>Donald</td>
<td>M</td>
<td>A</td>
<td>Eng.</td>
</tr>
<tr>
<td>Josh</td>
<td>M</td>
<td>A</td>
<td>Eng.</td>
</tr>
<tr>
<td>Thomas</td>
<td>M</td>
<td>B</td>
<td>Eng.</td>
</tr>
<tr>
<td>Eleanor</td>
<td>F</td>
<td>B</td>
<td>Eng.</td>
</tr>
</tbody>
</table>

**Table 3.2.** Participants from *M&I* course.

<table>
<thead>
<tr>
<th>Pseudonym</th>
<th>Gender</th>
<th>Grade</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martin</td>
<td>M</td>
<td>A</td>
<td>Comp. sci.</td>
</tr>
<tr>
<td>Denise</td>
<td>F</td>
<td>B</td>
<td>Physics</td>
</tr>
<tr>
<td>Zach</td>
<td>M</td>
<td>A</td>
<td>Comp. sci.</td>
</tr>
<tr>
<td>Carter</td>
<td>M</td>
<td>A</td>
<td>Comp. sci.</td>
</tr>
<tr>
<td>Geoff</td>
<td>M</td>
<td>B</td>
<td>Eng.</td>
</tr>
</tbody>
</table>
The two courses differed from one another in both course content and pedagogy. The topics in the M&I course were the momentum principle and forces, the energy principle, analyzing multiparticle systems, the angular momentum principle, and entropy and statistical mechanics. As mentioned in chapter 1, these topics are developed using the recurring themes of the atomic nature of matter, making connections between microscopic and macroscopic behavior, and the physical modeling of complicated systems. The first two-thirds of the traditional course covered classical mechanics, namely kinematics, Newton's laws, work and energy, gravitation, and momentum. The last third of the course covered heat and thermodynamics.

Particularly important to this study are the differences between the two courses in the presentation of fundamental physical principles. Students in the M&I course are taught three mechanical principles that are universal and can always be applied to any system. The first is the momentum principle, represented by the equation

$$\Delta \mathbf{p}_{\text{total}} = \mathbf{F}_{\text{net}} \Delta t$$

which says the change in the total momentum of a system, over a short time interval, is equal to the net force on the system times the short time interval. The momentum principle, used with the definition of momentum and the relation between position and velocity, can predict the trajectories of particles into the future. It can also be used to deduce what forces are required to give a particular known constrained motion to a particle. To predict motion, the momentum principle requires two supporting equations. The first of these is the definition of momentum, which at speeds low compared to the speed of light is

$$\mathbf{p} = m \mathbf{v}$$

that is, the momentum is the mass of an object times its velocity. For systems that contain many particles, the total momentum is the total mass times the velocity of the system's center of mass. The other equation used is
the definition of velocity $\mathbf{v}$, written in terms of the change in the vector position $\mathbf{r}$ of an object over a short time interval $t$: $\Delta \mathbf{r} = \mathbf{v} \Delta t$. The momentum principle is introduced almost immediately in the *Matter & Interactions* curriculum, and it is used as the primary principle upon which others are based.

The second major mechanical principle is the energy principle, represented by the equation $\Delta E_{\text{system}} = W + Q$, which says the change in energy of a system is equal to the (macroscopic) work $W$ plus the microscopic work (also called heat) done on the system. The energy of the system can be broken into a number of different terms, including kinetic energy (associated with the motion of particles within the system) and potential energy (associated with interactions within the system). The energy principle is used to relate different states of a system and predict states after a process, but it gives no information on time or duration of processes.

A third principle, the angular momentum principle, was also taught to *M&I* students, but the interview sessions took place just before the participants from the *M&I* section were introduced to it. Angular momentum was not taught in the traditional physics course at Carnegie Mellon during the semester the interviews took place.

The students in the traditional course were taught the same underlying mechanical principles, but they often appeared in different forms at different times during the course, and were given different levels of emphasis. In the traditional course, students were first introduced to the concepts of position, velocity, and acceleration. They then studied both one and two dimensional motion of objects undergoing constant acceleration, and made extensive use of special-case kinematics formulas that describe this motion. Constant acceleration equations were not emphasized in the *M&I* course.
Newton's second law in the form of $\mathbf{F}_{\text{net}} = m\mathbf{a}$ was next introduced in the traditional course, but it was almost exclusively applied to situations where the constrained motion of objects is known, not situations where the future motion of an object could be predicted from given force laws. Following this, energy methods were introduced in the form of the "work-kintic energy" theorem, $\Delta K = W$, which only applies to a single particle. Later, energy methods were developed by introducing conservation of energy in situations with gravitational or elastic potential energy. Finally, momentum was introduced, along with the impulse-momentum theorem, $\mathbf{J} = \mathbf{p}_f - \mathbf{p}_i$, where the impulse is $\mathbf{J} = \mathbf{F}_{\text{net}} \Delta t$. However, it is not clear how much practice the traditional course students had in using the theorem to solve problems where they had to update momentum given a force. Typically in the traditional introductory physics course, momentum is used far more often in collision problems, where the net external impulse to the system is zero and the total momentum of the system stays constant, than in problems involving the prediction of trajectories.

As for the logistics of the two courses, students in the M&I course met in lecture 3 hours per week and in recitation 2 hours per week. In the recitations, students did a number of different activities, including group problem solving, computer modeling using VPython, and occasionally, short, informal experiments. The traditional course consisted of a lecture that met 3 hours per week and recitation sections, consisting of approximately 15-25 students, that met two hours per week. Typically in recitations, teaching assistants worked example problems and answered students' questions. There was no laboratory component to the course.
3.3 Procedure

Each participant met with me individually and privately for an experimental session. During the session, the participant was instructed to solve a series of physics problems to the best of his or her ability. Each of the problems involved predicting the position of an object at a later time. The problem statements, as well as the instructions given to participants at the beginning of the session, are listed in section 7.2 of the Appendix. In problem 1, a satellite orbiting the earth (see figure 3.1) has a rocket engine which fires for 10 minutes producing a 400 N force in the direction of the satellite's velocity; participants were asked to predict its position after 15 hours. In problem 2, a ball kicked from the ground has a known initial velocity; participants had to find its position 0.90 seconds later while taking the force of air resistance into account. Problem 3 deals with a block on a spring attached to a central post (see figure 3.2). The block is free to move in two dimensions on a surface with negligible friction, and participants were asked to predict its position 6 seconds later, given its initial velocity. In all of the problems, the principles of mechanics that are taught in all of the introductory courses, especially Newton's Second Law, can be applied. However, these problems cannot be solved completely through analytic methods; a complete solution requires iterative use of the momentum principle. Participants were not told this, however; they were only told that the problems were difficult, and that they should not expect to solve all problems completely. This was not only to reduce stress on the participant, but also to reduce the tendency of the student to guess or invent invalid solution methods. Section 7.2 of the Appendix gives examples of VPython programs that solve each of these problems.
Participants were provided dry-erase whiteboards and markers to write out the solutions to problems. A scientific calculator, a graphing calculator, and a computer were also available; the computer monitor and keyboard sat on the participant's desk next to the whiteboard. Participants were told they were allowed to use the technology as they wished. They were also provided the textbook from their respective course and told that they could consult it at any time. Participants were instructed to "think aloud" while working, saying anything that comes to mind in the process of solving the problem. Participants' solutions were video and audio recorded. The video camera was placed
directly above the whiteboard to get a clear shot of participants' solutions to problems as they unfolded.

At the end of the experimental session, after the participants had completed all of the problems, they were asked follow-up questions about their solutions. Each participant was asked to rate his or her satisfaction with his or her approach to each problem, and at times, participants were asked to discuss solutions to certain problems in more detail where their reasoning was unclear or especially interesting. Lastly, participants were debriefed on their performance; this included a discussion of those problems that may have caused trouble, and a brief introduction to the iterative use of the momentum principle for those unfamiliar with it.

3.4 Analysis

All of the verbal protocols were transcribed. The videotaped sessions were transferred to recordable DVD; this allowed random access to any point in the sessions. The analysis of the verbal and video data was primarily qualitative. Quantitative analysis was held to the level of counting the number of problem instances (a single student's solution to a single problem counts as one instance) where certain features, to be discussed below, occurred. The data from this study is extremely rich and detailed, and lends itself to further analysis with regard to students' problem solving schemas, the conceptual and mathematical difficulties while solving problems, their use of prior knowledge, and their ability to evaluate their own problem solving performance. These issues fall outside the scope of this dissertation, however.
I will discuss in more detail the analysis of students' use of computational and iterative solution attempts in the next section. In section 3.6, I will discuss the analysis of the protocols for differences in solution approaches between the two groups of students.

3.5 Use of computation, iteration, and integration

The question of whether or not students wrote a computer program or used an iterative approach is simple enough to answer. The question of whether they thought such a solution was valid is more complicated to answer. To gain insight into this, I examined in detail the steps of thought and analysis students took that led to their choice to write a computer program. Occasionally, students made specific comments about their attitudes toward computer modeling during the protocols. Students who did not make use of computer modeling were asked at the end of the session if they believed a computer could help with any of the problems. Their answers at times revealed their beliefs about the legitimacy and ease of computer modeling. These types of responses are less valid, but they give some insight into their reasons for their decisions. I will not spend time discussing specific difficulties with programming; that question will be explored in chapter 4. None of the students from the traditional group used iterative methods, but some tried to deal with the changing forces in the problems through integration, although not necessarily of a relevant quantity. I will discuss these attempts in section 3.5.2. Table 3.3 summarizes the use of computation or iteration and integration by both groups of students.
Table 3.3. Number of solution attempts where computation, iteration, and integration were invoked.

<table>
<thead>
<tr>
<th></th>
<th>Traditional</th>
<th></th>
<th></th>
<th></th>
<th>Traditional</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p. 1</td>
<td>p. 2</td>
<td>p. 3</td>
<td>Total</td>
<td>p. 1</td>
<td>p. 2</td>
<td>p. 3</td>
<td>Total</td>
</tr>
<tr>
<td>Iteration or computation</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Integration mentioned or attempted</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3.5.1 Computation and iteration methods in the M&I group

As discussed in section 1.2, students in the M&I course were taught how to iteratively apply the momentum principle as a way to predict the motion of a particle. The iterative algorithm consisted of the following basic steps:

- Find the initial position \( \mathbf{r} \) and momentum \( \mathbf{p} \) of the particle.
- Use an the appropriate force laws and the superposition principle to calculate the net vector force \( \mathbf{F}_{\text{net}} \) on the particle.
- Using \( \Delta \mathbf{p} = \mathbf{F}_{\text{net}} \Delta t \), calculate the new momentum of each the particle a short time later \( \Delta t \) later.
- Using \( \mathbf{v} = \mathbf{p} / m \) (provided the speed is low compared to the speed of light), where \( m \) is the particle's mass, and \( \Delta \mathbf{r} = \mathbf{v} \Delta t \), calculate the new position of the particle \( \Delta t \) later.
- Repeat for as many time-steps as desired.

Students in the course practiced applying this algorithm in two ways. First, they worked exercises where they had apply this algorithm by hand to systems over one or two time steps. The second and most extensive use of this algorithm occurred when students were assigned to write computer programs in VPython that modeled the motion of particles in different systems.
Despite the fact that all of the problems involved predicting the motion of an object, and despite the presence of a computer in the experiment setting, only two of the five students from the M&I group, Martin and Carter, chose to write a program to solve the problems. No other M&I student used a computer, nor mentioned the possibility of using one during any of the problems. One student, Zach, did apply an iterative application of the Newton's second law by hand, though incompletely, to problem 2 (projectile motion with air resistance).

Martin and Carter ended up writing VPython programs to solve all three problems. It was not always their first choice of solution method; often they would briefly consider using a computer, but then try analytical means for some time before finally deciding to write a program. For example, for problem 1, Martin originally wanted to pursue energy methods, but realized that they would be insufficient for specifying an exact final position, stating:

So I'm thinking I want to approach this in terms of the energy of the system. And that will give me the radius away from the earth but it won't necessarily give me a position.

He then recognized an important aspect of the problem, and considered using a computer, saying:

The difficult part is the changing acceleration as a result of the earth's pull. It might actually be easiest to map or model this with a computer, but with a 15 hour time span to model I'm not sure the model is going to finish calculating in time.

The last statement is interesting in that it may indicate a misunderstanding of the nature of simulated time in a program. With an appropriate choice of time step, a computer model need not take an inordinate amount of time to simulate the motion of a system that has a large time scale, such as hours, months, or years. In fact, Martin had written in the M&I course a computer program to model the motion of the earth's one-year orbit around
the sun; when run, such a program only takes a few seconds to plot an orbit. So it is unclear why Martin seemed to believe 15 simulated hours would take a long time. It may have been just a slip: when Martin did finally write the program, he chose an appropriate time step and made no mention of a problem with the length of simulated time.

After mentioning the programming possibility, Martin started an energy analysis, stating that he wanted to find the "bounds on the state." Figure 3.3 shows Martin's work. He defined the system to be the earth and the satellite together, and he wrote that the changes in the energy and the momentum of the system are zero. He calculated the initial momentum of the system, and then wrote an equation for the initial energy of the system. He correctly realized that the energy of the system consists of the kinetic energy of the satellite, the gravitational potential energy, and the internal chemical energy of the firing rocket. After calculating the initial gravitational and kinetic energies, he turned his attention to the chemical energy.
The chemical energy gave Martin some trouble. He realized that he could not find the initial or final values of it, but only the change in it. The change in chemical energy in this problem is in fact numerically equal to the work done by the rocket force on the satellite, which cannot be calculated without knowing the distance over which that force acts. Martin realized this, saying:

I'm really interested in the change in chemical energy because that will be converted into some other form, so I don't know the initial chemical energy or the final kinetic energy, but I can find the change. The change is going to be equal to the force exerted by, hmm…. Ooh, that's a rough one. Delta, change in energy equals work done, I could consider the system the satellite but I don't know the distance the satellite is traveling through. So… I can say that in this model more or less, what I'm really interested in is more the change in the kinetic energy of the satellite and the change in chemical energy so I'm going to assume that the thrust exerted by the engines more or less changes the kinetic energy of the rocket.
He went on to calculate the "thrust" by multiplying the rocket force (400 N) by the duration of the force (600 s), but by checking units he realized the the quantity had units of impulse, not energy. At this point, he began to "feel like that's as far as I can go with this problem," but he continued by drawing a picture of the orbit (see figure 3.4). He mentioned that knowing the final kinetic energy of the satellite would help his analysis, but after this he finally decided to use the computer, saying:

I know how the forces are interacting here, so I'm going to go ahead and model the problem using Vpython.

His work up to this point took about 13 minutes.

Martin much more quickly decided to use a computer for problems 2 and 3. After reading problem 2, he said:

I'm trying to decide if I want to go straight into the math, or simply model in VPython, because right off the bat, I, I have, I already have a picture in my head of the way the forces are interacting on the uh, the ball.

He drew a diagram of the ball, and then said:

And using VPython I can easily iterate through .9 seconds of time modeling the g and air resistance forces acting on the ball...So I'm going to do that.
In problem 3, after writing constants a drawing a diagram of the block on the spring, he said:

Once again, I find this to be something that might be tricky to model out with the math alone. But the VPython modeling software would handle this relatively easily and give me an estimate to sufficient sig figs to solve the problem.

For all three problems, Martin wrote programs in a very straightforward way, with few errors beside simple syntax errors, which he easily corrected.

In problem 1, Carter at first considered writing a computer program, but then said, "I'd rather try and do it this way first," meaning analytically, apparently. He thought about doing an energy analysis but quickly ruled it out and decided to use the momentum principle instead, saying:

We can at least estimate...Let's see, change in momentum over change in time is equal to the force exerted. So this'll be an estimate of the change in momentum using the 400 newton force.

He first calculated a change in momentum of the satellite, but only used the 400 N rocket force in this calculation (middle of figure 3.5). He then realized that this was not all the forces, so he calculated the initial gravitational force and added it to the rocket force to find a net force (right side of figure 3.5). Although it is not clear, Carter's protocols suggest that he may have tried to approximate the satellite's velocity as constant over the first 600 s. His next step was to calculate an initial velocity and then find a distance by multiplying this velocity by the 600 s time. When he saw that that the distance moved was $1.5 \times 10^6$ m, he said "it's quite a bit compared to r, hmm...." Apparently, he believed this distance was too large compared to the distance from the satellite to earth's center to be a good approximation (even though it was less than a tenth of orbital radius). It was at this point he decided to write a program, 12 minutes after starting.
For problem 2, Carter again considered using the computer almost immediately after reading the problem and drawing a diagram, saying:

So once again it's more tempting to try computer prob--ah, to try and do it on the computer but going to start by figuring out on [the whiteboard].

However, after drawing a free body diagram and writing an expression for the vector net force, he realized the key feature of the problem which lent itself to computational methods:

All right so, hm, since the force is constantly changing due to the velocity constantly changing I'm going to write another computer program.

For problem 3, Carter considered and just as quickly dismissed energy methods. He then wrote the momentum principle and drew a free body diagram of the forces acting on the block. At this point, he said "I think I'm going to end up writing a computer program again," but he continued to do some more analysis on the whiteboard. For the
next few minutes, he calculated the initial length of the spring, and defined variables "l"
and "s" to stand for the length and stretch of the spring, before going on to write the
program.

Zach's approach to problem 2 was quite interesting. After reading the problem
and writing down relevant constants, Zach said:

So we're supposed to find it after .9 seconds where will the ball be. Now the speed will keep on
decreasing so have to take a reasonable time step.

His mention of time step suggests an iterative approach, as would be performed in a
computer program. After calculating the initial velocity components, he came back to the
issue of time step:

So now I say in time, what do we want to take as a reasonable time step? Point one [0.1], keeps on
going? We can try, we take .1 seconds as my time step.

He then drew a curved path, broken into discrete steps, and wrote "dt = 0.1 sec". (See
figure 3.6.)

Using the given formulas and his calculated initial velocity components, Zach then
calculated the values of the air resistance force components. At this point, he began to
use these force components to calculate new velocities via an iterative approach. It is
striking that he derived his iterative approach not from the forms of the momentum principle that were emphasized in the M&l course, namely $\Delta \mathbf{p} = \mathbf{F}_{\text{net}} \Delta t$ or even

$$
\frac{d\mathbf{p}}{dt} = \mathbf{F}_{\text{net}}.
$$

Rather, he starts with Newton's second law in the form $\mathbf{F} = m\mathbf{a}$, from which he derived

$$
F_x = \frac{m(v_{x2} - v_{x1})}{t}.
$$

(see lower left corner of Zach's board in figure 3.7), where $t$ refers to his time step. Using this, he solved for his new x-component of velocity, $v_{x2}$, and wrote:

$$
\frac{F_x t}{m} + v_{x1} = v_{x2}.
$$

(See top of board in figure 3.7.)
Next, Zach became concerned that a time step of 0.1 seconds would require 9 iterations, which is a lot of tedious calculations to do by hand. He briefly thought about finding a "general solution," probably meaning an analytic approach that could give a final answer without iteration. He stated:

Time step, if I take .1 as the time step, then I have to do it 9 times, or can I find a general solution to that?

He finally decided to change his time step, first to 0.2 seconds, then finally 0.3 seconds, saying:

More reasonable to take it as point 2...3.

Though he did not say it directly, this was probably to reduce the number of steps from nine to just three. While 0.3 seconds for a time step would not give a very accurate answer, it is a much better approximation to break the motion into three steps than to simply use the initial velocity and force to find a final position in one step. Zach, however, did not follow the full iterative procedure correctly: he did not calculate a new position for the ball at the end of each time step. Instead, he found new velocity components at the end of each time step, and then used them to find an average velocity. He then used this velocity to calculate a new x and y positions by simply multiplying the average velocity components by the total time of 0.9 seconds.

Zach's approach to problem 2 is unexpected for several reasons. While it is the case that students in the M&I course solve some problems by applying the discrete form of the momentum principle over one or two steps, they rarely apply it by hand to a lengthy trajectory, so it is unusual that Zach would think to apply it here. Furthermore, it is odd that he derives the approach from $\vec{F} = m\vec{a}$. Since this formula is rarely mentioned in the M&I class, Zach is probably invoking something he learned in a previous physics
course, perhaps from high school. But it is extremely unlikely that a previous physics
course would teach how to apply Newton's second law iteratively, so this may be a case
of Zach combining new approaches from M&I with prior knowledge. It's not clear why,
given Zach's iterative approach, that he did not use a computer. It's possible he did not
realize that his iterative approach and the computer algorithm is essentially the same.
Even if he did, he may not have used a computer out of personal discomfort with his
programming abilities, or because of context: written problems require written solutions.
At the end of the session, when asked if a computer could help with problem 2, Zach
replied, "Yes, because it is something in which you keep on updating the time step. So, it
would be pretty convenient to do that." So it appears in retrospect, he realized that it was
possible, but it did not occur to him during his solution. It is also not clear what aspects
of the problem suggested to him to make use of this approach. He did not use it problem
1, nor did he continue to use it in problem 3; in both cases he tried to use energy methods.

Denise and Geoff did not make use of computational or iterative methods in any
of the problems. At the end of the session, they were asked if a computer would have
helped in any of the problems. Both expressed discomfort with computer modeling. For
example, in post-session debriefing, Denise stated that she "was not a big fan of computer
program writing," even though she had previously taken a programming course and had
done well in it. She thought that a computer might help "numerically," but she felt she
was "more of a theoretical person" and liked "solving things by hand." The biggest
problem to her was "figuring out the algorithm." Geoff thought that a computer program
"would definitely help if you were good at programming." So it appears that
unfamiliarity with the language and rules of programming may hold students back from choosing it as a legitimate approach.

3.5.2 *Integration use in the traditional group*

No students from the traditional group mentioned the possibility of using a computer to solve any of the problems, nor did they attempt an iterative application of the momentum principle for any problem. This is expected, since these students were not taught explicity taught computational or iterative methods in class. But it is noteworthy as further evidence that iteratively applying Newton's second law is not a problem-solving method that students from a traditional physics course can simply invent, even though it is based on principles and equations that students in such a course use quite often.

The critical problematic feature of these problems is that the net force on an object, and therefore its acceleration, is changing with time. Four of the six traditional course students were at least aware of this issue as evidenced by mentioning the possibility of doing an integral, or by attempting to set one up. In principle, an integral of an acceleration function over time would give a velocity, which in turn could be integrated over time to find a final position. For these problems, such integrals could not be solved analytically, but properly setting one up might at least demonstrate an understanding of how the momentum principle can be used to predict the motion of objects. Those students in the traditional course who did write integrals, however, were often confused about what exactly to integrate, including the function, integration variable, and limits. They did not use the momentum principle or Newton's second law to set up an integral of force over time, but rather attempted to integrate force over
distance in a work integral. This could be because in class and in their textbook these students were exposed to the idea of integrating a force to get a function for work, often demonstrated in the work done by gravitational or spring forces. It is most likely the case that the traditional students had not previously done any problems involving integrating a changing force over time.

In problem 2, Josh realized that the forces would change with time:

OK if there were no air resistance in this problem, we could predict the position pretty well using simple projectile motion, however, need to figure out the, acceleration, the force and the accelerations caused by the air resistance, which are of course changing as the speed changes. I don't remember this so, um, looking up how forces change in Chapter 4.

Chapter 4 of the Young and Freedman text deals primarily with constant forces, so Josh did not find any useful information. A short time later, he thought about using work and energy, saying that the "work done by air resistance is gonna equal the change in kinetic energy." He then searches chapter 6 of the text, which introduces work and energy, saying, "One equation I'm looking for with a varying, varying force, something like an integral or something." He found the general definition of work, the integral of the force over a distance, and wrote it down (see figure 3.8, bottom left). Josh next set up a work integral using the air resistance force (see figure 3.8, top right). He soon began to see that the major problem with this approach—the distance over which the force is being integrated is also the quantity that needs to be found to solve the problem. He said, "This [the work integral] doesn't work, because I don't know where the, uh, thing ends up." In a different approach, he dropped the integral and just multiplies the force by the distance (see figure 3.9), but again realized the same problem, saying:

I guess I'll just multiply all this by some x, the distance we end up traveling. Which is also what we're trying to find out in the end so, this isn't good.
Notice also that he apparently tried to find an "x-component" of the work. Coming back to the integral, he again said:

To do it this way, I need to know the distance we travel in the end, and the velocities are changing as we go, so I don't see how this is going to work at all.

Soon after, not knowing what step to take next, Josh gave up on this problem.

Figure 3.8. Josh, problem 2.

Figure 3.9. Josh, problem 2. Work "in x direction" is the force times the distance.
Jerome, like Josh, also tried to use a work integral to solve problem 2. After setting up an integral over position of the air resistance force (see figure 3.10), he realized that his function was not in terms of the integration variable, saying:

Trying to figure out how to integrate this, have to do it with respect to x, so you need to solve--you need to find out what $v_y$ is in terms of x.

Note the use of "$W_x\$", indicating the work done in the "x direction", which shows a misunderstanding of the scalar nature of work and energy. He erased this equation, saying:

Actually, can't use this equation, because it's along a curved path because of gravity, not along a straight path. So need to use the equation for work along a curved path.

Even though he made this false distinction between the two cases, he ended up writing essentially the same equation again; in fact, it's even more clear that he broke the work into directional "components." (see figure 3.11). Later, he even mixed energy terms and constant acceleration kinematics terms in an equation, but he abandoned this approach. Ultimately, his final idea was that the individual "x" and "y" work integrals could be added to the kinetic energy in "each direction"; the y equation also includes the work done by gravity through an unknown distance "d" (see figure 3.12). Provided that the integrals could be solved, Jerome claimed that these equations could then yield the final position. He gave up at this point.
Figure 3.10. Jerome, problem 2.

Figure 3.11. Jerome, problem 2. Note "x" and "y directions" for work.
Thomas in problem 2 also talked about using a work integral, and wrote a general one (omitting the differential, see figure 3.13). After this, he struggled with where to go next. He thought about the formula "$kx^2/2$", but quickly realized no spring was involved. After thinking more about the direction of the force and the displacement, he wrote a general work-energy equation, but then realized that the x-component of the final velocity is unknown. He had first thought that he could treat the x-component of velocity as constant and then multiply it by the total time to get an x-position, but realized it would not work. After thinking some more, he wondered if he could just ignore air resistance, but said "that's not what they want," and "it would give me a rough estimate, but that's not going to be too good." He finally gave up, unable to solve the problem.
Donald in problem 1 also tried to set up an integral, but he was not clear on what exactly to integrate. Concerning the gravitational force, he said, "So that's obviously gonna change over time which means this is gonna be an integral." After drawing the gravitational and rocket forces on the object, he chose coordinate axes that were parallel and perpendicular to the initial direction of motion. He then wrote an integral of the parallel component of the net force (the gravitational force component plus the rocket force). He wrote limits of integration that were values of time (from 0 to 600 seconds), but then said that the integral was over the "radius", which led him to write "dr" as the differential and erase the time limits, asking "Why am I doing that?" He then rewrote the time limits and said that it was "proof that I'm not quite sure what I'm doing at all on this problem" (see figure 3.14). Next he wrote an integral of the gravitational force expression with respect to the variable "r". The initial limit was $4.0 \times 10^7$ m times the cosine of the angle between the perpendicular and the direction of motion; the end limit
was an unknown distance "r". Donald was unsure of what to do with this, and instead he began to think about momentum methods; he wrote "400 Δt" to represent the impulse—he was clearly thinking about the rocket force alone. He then tried thinking about momentum conservation and wrote "p_i - p_f", but ultimately he decided that he did not know how to finish the problem (see figure 3.15). At no point did he ever mention that the integrals he wrote were calculations of work, or of impulse.

Figure 3.14. Donald, problem 1.
Figure 3.15. Donald, problem 1. Note integral of gravitational force and an impulse calculation in lower right.

3.6 Differences in solution approaches

We have seen that some students from the M&I course used computational or iterative solution methods, though perhaps not as many as expected or desired. Closer examination of the protocols can give insight into another aspect of the effects of the M&I course. As discussed in chapter 1, the M&I curriculum emphasizes not only computer modeling but also fundamental principles, and both of these may affect students' problem solving approaches. It is conceivable that M&I students will be more likely to derive a solution from a fundamental physics principle rather than try to match the problem to a known solution of a different problem. There are many complex aspects
to student problem solving, but I will limit my examination to the principles and
equations that the students used in their main solution approaches.

Certain features of the students' protocols and problem solving steps can serve as
evidence for characterizing solution approaches as ones which rely more on fundamental
principles or more on matching to known solution types. To find these features, these
questions were asked while examining the protocols:

1. *Did students make use of the momentum principle?* This principle is the only
   possible starting point for successfully completing these problems. Energy
   methods may be able to provide bounds on possible distances, but they cannot
   predict final vector positions in these problems. That is because the problems
   involve motion over specified times, not motions over specified distances, or
   motions with known final velocities. If student did use the momentum
   principle, even though they may not have used a computer, it at least shows a
   starting point whereby students could be persuaded or taught to use
   computational methods (which happened several times during the pilot).

2. *How often did students make use of pre-derived, special-case equations or
   solutions?* It is possible that students who start the analysis of a new problem
   from a fundamental physics principle may be more likely to reason from it
   and derive secondary formulas and results. Others may resort to formulas or
   methods that match a solution to a particular problem type, even when those
   formulas are invalid for the system being examined. The two most important
   examples in this study are equations of motion for objects undergoing
   constant acceleration and equations and methods for circular motion; neither
of these situations occurs in any of the problems used in the study. Use of these equations could indicate a lack of understanding of their validity conditions and of how they are derived from fundamental principles.

3. **How often did they resort to looking in the book for examples or formulas?**

   Along the same lines as above, students who spend more time looking through the book for formulas and examples may be using an approach that relies more on problem matching than on starting from fundamental principles.

Another minor issue of interest that I will examine, not directly related to fundamental principles, is whether the students proceeded through the problems to find what they believed to be a satisfactory solution (even if it was incorrect), or gave up along the way. Given that  

Given that M&I students have practiced solving many complicated problems, they might have a greater expectation that solutions to these problems are possible to reach with the tools and methods that they know.

Tables 3.4 and 3.5 summarize quantitatively the problem instances where the above major features are found. Both tables display the same data, where table 3.4 is organized by student group, while table 3.5 is organized by problem.

<table>
<thead>
<tr>
<th>Solution path feature</th>
<th>Traditional</th>
<th></th>
<th></th>
<th></th>
<th>M&amp;I</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p. 1</td>
<td>p. 2</td>
<td>p. 3</td>
<td>Total</td>
<td>p. 1</td>
<td>p. 2</td>
<td>p. 3</td>
<td>Total</td>
</tr>
<tr>
<td>Used ( \Delta p_{\text{total}} = F_{\text{net}} \Delta t ) (but no iteration or computer model)</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Used ( F_{\text{net}} = ma )</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>11</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( \frac{1}{2} at^2 ) equations</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Circular motion equations or methods</td>
<td>4</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Book used for examples or equations</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 3.4.** Number of solution attempts with specific features, by group.
Table 3.5. Number of solution attempts with specific features, by problem.

<table>
<thead>
<tr>
<th>Solution path feature</th>
<th>p.1</th>
<th>p.2</th>
<th>p.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Used $\Delta\vec{p}<em>{total} = \vec{F}</em>{net} \Delta t$ (but no iteration or computer model)</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Used $\vec{F}_{net} = m\vec{a}$</td>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$\frac{1}{2} at^2$ equations</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Circular motion equations or methods</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Book used for examples or equations</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
</tbody>
</table>

3.6.1 Use of momentum methods

While students from both courses often explored different solution paths using different equations or principles within individual problems, most of them made use of the momentum principle in some form. In thirteen out of the fifteen solution attempts, including the six computational solutions, the M&I students ultimately chose a method that made use of the momentum principle in the form $\Delta\vec{p}_{total} = \vec{F}_{net} \Delta t$ or $d\vec{p}_{total}/dt = \vec{F}_{net}$.

As mentioned in section 3.5.1, Zach derived his iterative approach to problem 2 from $\vec{F}_{net} = m\vec{a}$, while Denise in problem 3 did not use any form of the momentum principle. In the seven solution attempts, students applied the momentum principle over a single time step that spanned from $t=0$ to the final time mentioned in the problem to find an updated momentum. In five of these seven attempts, they then used the new momentum to find a new velocity, which they then used to find a final position in some way (Zach in problem 3 gave up before updating position, while I moved Denise along in problem 2 due to time constraints). Of these five, Geoff in all three problems found the position by using the definition of velocity—essentially, the position was calculated as if the velocity stayed constant over the entire time period. Denise in problem one used the "$at^2/2$" formula to find the position, and Zach in problem 1 used energy conservation. So
in total, including those who used computation or iteration, 10 out of 15 solution attempts involved the "canonical" steps of predicting a new position by using $\Delta \vec{p}_{\text{total}} = \vec{F}_{\text{net}} \Delta t$ (or $v = v_0 + (F/m)\Delta t$ in Zach's case) to find a new momentum, then using $\Delta \vec{r} = \vec{v} \Delta t$ to find new position.

The students from the traditional course also made wide use of the momentum principle, where it appeared in 13 of 18 solution attempts. The biggest difference from the $M&I$ group was that they usually used it the form that was most familiar to them (as discussed in section 3.2), namely, Newton's second law, or $\vec{F}_{\text{net}} = m\vec{a}$, which appeared in 11 of the 18 solution attempts. Only one student from the traditional course (Donald), in two problems, made use of the momentum principle in the form $\Delta \vec{p}_{\text{total}} = \vec{F}_{\text{net}} \Delta t$ (known to him as the impulse-momentum theorem) as his ultimate solution method. This is not surprising considering the content of the two courses; $\Delta \vec{p}_{\text{total}} = \vec{F}_{\text{net}} \Delta t$ is emphasized from the very beginning of the $M&I$ course, while $\vec{F}_{\text{net}} = m\vec{a}$ is far more prevalent in the traditional course. However, there does seem to be a wider range of disparate ways that the momentum principle was used to find a final position (if at all) among the traditional students than the $M&I$ students. Consider the following cases:

- In two cases, (Donald, problem 1; Thomas, problem 2) $\vec{F}_{\text{net}} = m\vec{a}$ was written but never used. In another (Josh, problem 1) $\vec{F}_{\text{net}} = m\vec{a}$ was used to find an acceleration, but the acceleration was never used (the student gave up shortly thereafter).

- Two solution attempts (Sam, problem 3, and Thomas, problem 1) simply set the acceleration in $\vec{F}_{\text{net}} = m\vec{a}$ equal to $v^2/r$, which gives the magnitude of the
acceleration of an object in circular motion. They then set the force equal to the gravitational force and solved for a speed.

- Two more (Jerome and Sam in problem 1) started off by finding an acceleration from \( \vec{F}_{\text{net}} = m\vec{a} \), which was then used in \( v = v_0 + at \) to find an updated velocity. However, circular motion equations and methods were used after this.

- Eleanor in problem 1 found an acceleration using \( \vec{F}_{\text{net}} = m\vec{a} \), then used an incorrect equation, \( v = v_0 + at^2/2 \), to find a new velocity. She then updated position using the definition of velocity; that is by using \( \Delta x = v_x \Delta t \) and \( \Delta y = v_y \Delta t \).

- In three cases (Eleanor, problems 2 and 3, and Sam, problem 2), an acceleration was found using \( \vec{F}_{\text{net}} = m\vec{a} \), which was then plugged into \( at^2/2 \) distance formulas to find a new position. For more details, see section 3.6.4.

This wide range of confused applications of \( \vec{F}_{\text{net}} = m\vec{a} \) suggests that the traditional students were not accustomed to using Newton's second law for the prediction of motion. The very form of the equation, which does not explicitly include time, even suggests an algebraic relationship between static variables, rather than time evolution. M&I students were far more used to applying the motion update algorithm that had been emphasized in their course, even those who only applied it over one large time step. Students in M&I had been taught that updating the momentum and then the position over one time step could be used to approximate the new position of an object, provided the time is short enough; none of the participants, however, ever mentioned they were making an approximation.
The steps of applying $\Delta \mathbf{p}_{\text{total}} = \mathbf{F}_{\text{net}} \Delta t$ followed by $\Delta \mathbf{v} = (\mathbf{p}/m) \Delta t$ to find an updated position were taught to M&I students as a general algorithm. Because of this, I will not treat this approach as an instance of using special case formulas, even though the assumptions of constant force or constant velocity are not justified in these problems. A few of the traditional course students followed steps that were functionally equivalent to all or part of this general algorithm. That is, they used $\mathbf{F}_{\text{net}} = m \mathbf{a}$ to find an acceleration, then used $\Delta \mathbf{v} = \mathbf{a} \Delta t$ to update velocity; as mentioned above, Jerome, Sam, and Eleanor did this in problem 1, though Eleanor made a mistake in the velocity formula. Eleanor went on to use $\Delta \mathbf{r} = \mathbf{v} \Delta t$ to update position. Giving these students the benefit of the doubt, I will not treat these approaches as evidence of using special case formulas. I will, however, count approaches where an acceleration was found and then plugged into a constant acceleration kinematics formula for distance, such as $x = a_x t^2 / 2 + v_{0x} t + x_0$, or $x = \sqrt{(v_x^2 - v_{0x}^2)/(2a_x)}$, as instances of special case formula use. This is not to say that using these formulas is somehow "worse" than updating momentum or velocity with $\Delta \mathbf{p}_{\text{total}} = \mathbf{F}_{\text{net}} \Delta t$ or $\Delta \mathbf{v} = \mathbf{a} \Delta t$ and updating position with $\Delta \mathbf{r} = \mathbf{v} \Delta t$, since the underlying assumptions are violated in all cases. It just reflects a qualitatively different approach to the problem. I will discuss students' use of special case kinematics formulas in more detail in section 3.6.4.

Even though most students started with the momentum principle, many students struggled with the details of using it to reach a solution. For example, in problem 1, Zach started with the momentum principle, and calculated a change in momentum of the satellite using $dp = F dt$, where $F$ was the force due to the rocket only—the gravitational
force was not added (see figure 3.16). This was a very common mistake for students of
both groups in problem 1. After getting this change in momentum, he divides by the
mass to get a change in velocity, and subtracts this from the initial velocity to get a new
velocity (see figure 3.17). (Zach's subtraction may have been caused by poor wording in
the problem statement. The problem stated that the satellite "fires its rocket in the
direction of motion." Some students may have interpreted the statement to mean that the
force's direction was opposite to the velocity's, even though the problem goes on to say
that "the rocket applies a 400 N force in the direction of the satellite's velocity." ) After
drawing several vector diagrams, Zach decided to use the energy principle (see figure
3.18). For the final kinetic energy term, he used the final speed that he calculated
through the momentum principle. He then calculated a final distance from earth, but was
unsure what to do after the additional 15 hours. He realized that gravity would affect the
satellite's motion, and he considered using the momentum principle and the gravitational
force (see figure 3.17, lower left), but he ultimately gave up, unsure of what to do next.
Figure 3.16. Zach, problem 1.

Figure 3.17. Zach, problem 1. Change in $v$ in lower right used to find new $v$. 
Figure 3.18. Zach, problem 1. Note energy principle, below diagram.

Figure 3.19. Zach, problem 1. Note momentum principle in lower left.
Donald was the only student from the traditional course who made use of the discrete form of the momentum principle to solve any of the problems. Though he did briefly invoke but abandon impulse and momentum in problem 1, he used it as the sole strategy in problem 2, and again invoked it in problem 3 as his last solution path. In problem 2, after studying the problem situation, he decided to use impulse-momentum methods, stating that he was influenced by the results of a recent test that he took that covered energy and momentum ("this is probably just because I just had that test and I just got it back and I did really poorly on it"). After writing the impulse momentum equation (see figure 3.20), he then dealt with the x-component and y-components separately. He used the initial x-component of momentum and added an impulse that was the x-component of the air resistance force multiplied by the total 0.9 second time of flight (see figure 3.21). He did the same for the y-component—he only included the air resistance force in his calculation of the y-component of momentum, and did not mention the gravitational force (see figure 3.22). Plugging the initial velocity components into the air resistance force formulas, he was able to calculate values for the final x and y components of the velocity after the 0.9 seconds. To find a final position, Donald made use of a constant acceleration kinematics formula. He incorrectly used the magnitudes of the initial and final velocities in this formula, however, instead of using it for the x and y-components of the position independently (see figure 3.23).
Figure 3.20. Donald, problem 2.

Figure 3.21. Donald, problem 2. Calculation of a new x-component of $v$ in lower right.
In problem 3, Donald started out by using work and energy, but was concerned with the fact that he did not know the final spring potential energy, because he did not have the final position of the block. This was, instead, what he was trying to solve for. He considered assuming the spring ends up in a relaxed state (thus giving a final potential of zero), but then he decided to change his approach. He said:
So this could be just following in the success of my last problem, but I really want to use momentum again, because I've got a direction, and even though the spring would give me potential energy and that seems to be the thing I would wanna do, I will have no idea where it ends up since I have to figure out where it actually decides to travel in that time.

At this point, Donald worked the problem using the same procedure he used in problem 2: update the momentum using the impulse-momentum theorem over one large time step, then use a constant acceleration kinematics formula to find the final displacement.

3.6.2 Special case formulas: circular motion

The $M&I$ students appeared less likely than the traditional course students to use special case formulas or methods that were inappropriate and invalid for the systems given in the problems. One example is the use of circular motion methods and equations. None of these problems involve cases of uniform circular motion. However, problem 1, which involves an orbit, and problem 3, which involves a radial spring force, have surface features that suggested circular motion methods to some students. In the traditional group, there were five problem instances among three students where circular motion methods were used. In the $M&I$ group, only one student (Denise) applied circular motion methods to one problem (problem 3).

The main sticking point among the students who used circular motion methods was what to do with the velocity, which was not tangential to a circular path. By definition, the velocity vector at any point along a trajectory is tangent to the trajectory, but students conceived situations where only a component of the velocity was tangent to the circular path, and another component was perpendicular to the path. Students either ignored this issue, or made invalid assumptions, in order to simplify the problem to one they could deal with. That this was much more prominent among the traditional course students likely stems from the fact that unlike the $M&I$ students, the students in the
traditional course were never exposed to problems involving orbits that were not circular. This is to be expected since the mathematics involved in dealing with elliptical or other non-circular motions is too complex for introductory students. However, through computer modeling, M&I students had dealt with several cases of non-circular motions, including elliptical and hyperbolic orbits. This probably made them less likely to assume circular motion in problems 1 and 3.

Jerome in problem 1, Sam in problem 1 and 3, and Thomas in problem 1 and 3 used circular motion methods as the main solution method. In problem 1, Jerome started off with a solution approach that was used by several other students: find an acceleration using $F=ma$, where $F$ is the rocket force alone, then use this acceleration in $v=v_0 + at$ to find a new velocity (see figure 3.24). At this point, Jerome searched the book for, as he put it, "equations for motion of satellites." Note the extreme specificity—he wanted to find equations that fit the surface features of the problem. Next he appeared bothered by the fact that this problem does not fit the expected behavior of a circular motion situation, because the velocity is not tangent to a circular path. He said, "I know how it would orbit if it was just, if the velocity was tangent to the circle, but it's going out 15 degrees." But he apparently believed that somehow, perhaps through the gravitational force, the satellite would settle into a circular orbit: "I could figure out the force of gravity on it until it reaches, orbit that's parallel to the earth." He found in the book the equation $v = \sqrt{GM / r}$, which is the speed of a object in a circular orbit around a planet of mass $M$, at an orbital radius of $r$, and he wondered how to use his velocity in this equation:

I think this should work using equation for $v$ to find out how high it is I'm not sure if you have to use the velocity, just the velocity that's par--, that's ah, tangent to the orbit or if you have to use the whole velocity.
Later, he decided to simply ignore the "radial" component of the velocity, saying:

I think, go with just ignoring the velocity away, think that gravit--the work done, that gravity will, uh, keep that from making a difference.

In other words, he justified away the key feature of the problem situation in order to make it fit the pattern of circular motion. Following this, he even abandons the use of the $v = \sqrt{GM/r}$ formula. He instead finds the distance traveled at the "tangential" speed by multiplying it by the 15 hours, then calculates what fraction of a circumference of a circle this arc-length represents (see figure 3.25).

![Figure 3.24. Jerome, problem 1.](image-url)
Figure 3.25. Jerome, problem 1. Notice circumference calculation at bottom, distance calculation at top right, and ratio of distance to circumference at lower right. The answer says, "0.628 of way around its orbit."

Sam solved problem 1 by first calculating an acceleration using $F=ma$, where $F$ is the 400 N rocket force. He then found a new velocity from $v=v_0 + at$. Using this new velocity, he sets the gravitational force equal to the mass times centripetal acceleration, thus implicitly indicating the motion is circular. Lastly he solved for the final orbital radius (which does not fully answer the question, since it does not give $x$ and $y$ coordinates).

Sam's solution to problem 3 is similar to Jerome's solution to problem 1 in that he realized the block has a radial component of velocity, but ignores it in order to fit the problem to the circular motion template. Realizing at first that the block is radially accelerating, he first set the spring force equal to the mass times the radial acceleration, using the total current length as the radius (see figure 3.26). He then drew the block taking a circular path, essentially holding the radius constant and ignoring any oscillatory motion (see figure 3.27). Reaching an impasse, he looked in the book for radial
acceleration and circular motion formulas. Finding what he wanted, he calculated the tangential component of the velocity to this circle, and used this component to find an angular speed. Treating this angular speed as constant, he multiplied it by time to find an angle in radians, and then multiplied the angle by the radius to find the circumferential distance the block moved on its circular path.

**Figure 3.26.** Sam, problem 3. "x₀" is the original spring length, "x" is the stretch.

**Figure 3.27.** Sam, problem 3.
Thomas initially drew a diagram of the situation in problem 1 where the path of the satellite was circular (see figure 3.28). Realizing that the velocity was not tangent to the path, he altered the diagram to make orbit a strange shape (see figure 3.29). He said, "I don't know why the velocity is not tangential," but then he noticed that the angle between the velocity and the radius was 105 degrees, not 90, and said, "maybe its orbit is a little bit different." Thomas was unsure on how to complete the problem; he calculated the gravitational force magnitude, and briefly thought about using work and energy. Ultimately, he calculated an orbital period based on the gravitational force, and stated that 400 newton rocket force would cause the period to decrease. At this point he gave up.

Figure 3.28. Thomas, problem 1.
Denise, from the M&I group, in problem 3 drew a diagram similar to Thomas's diagram from problem 1 (see figure 3.30). The block moves in a circular path despite having a radial velocity component. In solving the problem, Denise confused two different uses of the symbol omega. She first calculated the angular oscillatory frequency of the spring, based on the spring constant and the mass of the block, \( \omega = \frac{\sqrt{k}}{m} \). But she then believed this was the same as the angular frequency of rotation, \( \omega = \frac{2\pi}{T} \), where \( T \) is the period. Using this incorrect rotational angular frequency, she calculated the final angular position of the block.
3.6.3 Special case formulas: constant acceleration kinematics

The M&I students did not appear to use projectile motion equations as much as the traditional group did. Here I refer specifically to the formulas that give the position of an object undergoing constant acceleration as a function of time, or of initial and final velocities, such as \[ x = a_x t^2 / 2 + v_{0x} t + x_0, \] and \[ x = \sqrt{(v_{y0}^2 - v_{0x}^2)/(2a_x)}. \]

Three students from the traditional course made use of these formulas in their solutions in a total of five problem instances. Two of these instances were Donald's solutions to problems 2 and 3, previously discussed in section 3.6.2. Three of these instances were on problem 2, which dealt with projectile motion with air resistance; the surface features of this problem strongly suggested the use of projectile motion formulas. Among the M&I students, only one student, Denise, used these formulas in a single problem: again, problem 2. A second student, Geoff, did write down these formulas in both problem 2 and 3, but in each case abandoned them without using them in his solution.

Eleanor, from the traditional group, approached all of the problems by using \( F = ma \) to find an acceleration, then used this acceleration in constant acceleration (or
constant velocity) kinematics equations to find the final position of the moving object.

For example, in problem 2, she first calculated the x and y components of the initial velocity and plugged them into the given air resistance force equations to find the air resistance force components. She then began to use kinematics equations, whereby she found a value for the change in the y-component of position (see figure 3.31). Thinking about force again, she was concerned about how to put the gravitational force $mg$ and the air resistance force together, but decided to add the y-component of the air resistance force to the gravitational force. Using $F=ma$, where $F$ was the net force in the y direction that she had just found, she calculated an acceleration. She then considered placing this acceleration back into the kinematics equation, but she was troubled by the value of the acceleration (different from $9.8 \text{ m/s}^2$) and by not knowing the "v final in the x direction." Despite her belief that this was incorrect, she did just this, and found a new y value, which turned out to be a negative number. This led her to conclude that her answer was wrong since the ball "can't go into the ground." Saying that she had "no idea anymore," she then quit.
Figure 3.31. Eleanor, problem 2.

Sam (traditional group) followed a similar course in problem 2. He used the given formulas to calculate the air resistance components, and added the y-component of air resistance to the gravitational force for a net y-component of force. He then found the acceleration components by $F=ma$, and then turned to the textbook to find kinematics equations, saying:

So because we have the same information in both the x and y direction we can most likely use the same equation to find the positions, so let's refer to the book, force and kinematic equations, because I don't know them off the top of my head unfortunately.

And later:

Ok, so if we have acceleration of, uh if we have constant acceleration, um we can use kinematic equation and we also know initial velocity, time, and we want to know position... So, so now I found a good equation to use, we can use, in the books it is written as $x$ equals $x$ initial plus $v$ initial times time plus one half acceleration times time squared.

Notice that he talked about the equations being for constant acceleration, but he either did not realize that the acceleration in this case was not constant, or he did not explicitly state
an assumption of constant acceleration in order to simplify the problem. He finally plugged the accelerations, initial velocity components, and total time of flight into the kinematics equations to find a final position.

Denise was the only M&I student to use the constant acceleration distance formulas in her solution to problem 1. However, her approach was much less straightforward than those of Eleanor or Sam. Denise started out thinking about the two main fundamental principles that she had learned: the momentum principle and the energy principle, and wrote both of them down. She was unsure of which one to use, saying, "My mind is thinking in two directions, should I use energy or should I use momentum to figure this out?" She decided to use the energy principle and included the initial and final kinetic energy and gravitational potential energy terms (see figure 3.32). Writing this in more detail, she substitutes the definition of kinetic energy for the $K$ terms, and the definition of potential energy for the initial $U$ term (with the wrong sign, and $r$ incorrectly squared in the denominator). She did not realize that the internal energy of the satellite would also change, due to the firing rocket, but her other remarks indicate that she may not have know when the rocket was firing, or how the rocket force was applied. Noticing the unknown final velocity in the energy equation, she thought about using the momentum principle to compute it, but then she noticed that the final distance in the potential energy term was also unknown. At this point she began to "wonder if this is valid," referring possibly to her energy approach, but she then wrote the final potential energy term, and decided that this included the final position she was looking for. Next,
she rewrote the kinetic energies in terms of momentum instead of velocity (bottom of figure 3.32). But then, just before she began doing the calculations, she said:

Wait, what am I doing now? It's not changing direction, but it's changing momentum. We have change in momentum.

At this point, she focused on the momentum principle, and used it calculate a change in momentum, with the only force being the 400 N rocket force. (She did not include the gravitational force, even though she had used gravitational potential energy in her energy equation.) After calculating a distance based on the initial speed and a five hour time (which may have been prompted by a misreading of the problem), she began to treat the problem situation as if the satellite were moving in one dimension with constant acceleration. She found an acceleration by calculating $\frac{\Delta v}{\Delta t}$, where $\Delta v$ was the change
in momentum she had just found divided by the mass, and \( \Delta t \) was the 10 minute time over which the 400 N force acted. This gives the same result as using \( a = F/m \), where \( F \) is the rocket force, and she realized this, saying "Or I could have just divided the force by the mass and gotten the acceleration" (see figure 3.33). Next, she used constant acceleration kinematics to find a distance \( D \) that the satellite traveled during the 10 minutes. She found another distance \( d \), the distance that the satellite traveled after the rocket fired, that was found using a constant speed over 5 hours (perhaps a misreading of 15 hours in the problem). She added \( d \) and \( D \) to find a total distance (see figure 3.34). By the end she seemed confused, and wondered why she had switched principles, saying, "I'm sure I could do this a different way, why didn't I stick to my energy equations?" She also said:

I could double check that because I'm very not sure about this. I wonder if I could do it that way with my energy equations and compute this \( r \) [the distance from earth to the final position that she found].

While she wanted to continue, I moved her along to the next problem due to time constraints. Overall, Denise's approach did start with fundamental physical principles, but as time went on and confusion grew, she seemed to fall back on special-case formulas.

Figure 3.33. Denise, problem 1. Calculating an acceleration.
3.6.4 Textbook usage

*M&I* students looked in their textbook for help less often than the traditional students. Three students in a total of three problem instances looked in the textbook for a specific formula. (I am ignoring instances where students simply looked up the value of a constant; two such instances occurred among the *M&I* students.) This is in contrast to students from the traditional group, where four students in a total of ten problem instances looked for equations in the textbook. Occasionally, these students were not even sure what specific equation they were looking for; instead, they were just more generally browsing for anything to help them with the problem, including a worked example. Recall, for instance, Jerome's search for "equations for motion of satellites" in the textbook during problem 1. Another example is Josh, who made remarks when he
was working on problem 1 that clearly indicate he was trying to find a worked example similar to the problem:

So I'm going to consult the book and see if I can find anything useful. And I'm going to, think it's chapter 4 or 5, applications of Newton's laws, and I'm just going to look around for something that might be helpful, or try to find an example problem that's similar to this one with a force and direction changing. No not circular motion, or any of this. Oh, yeah but the gravity problems are in the back. Think it was like chapter 11. OK, motion of satellites, see if I can find a complicated problem like the one I have.

3.6.5 Problem completion

The M&I students were more likely than the traditional students to work a problem to completion, where completion is defined as reaching a numerical answer to the original question with which they were satisfied, despite its correctness. Five of the traditional students, in ten separate problem instances, decided to give up at the end of the problem without reaching a final answer, believing that they had no more solution options to pursue. In the M&I group, only one student (Zach) on one problem (problem 3) voluntarily gave up. On problem 1, Denise reached an impasse and moved on to the next problem, but wanted to come back and complete problem 1. She never did, due to time constraints. I also decided to tell Denise on problem 2 and Geoff on problem 2 to move on due to time constraints.

Josh, from the traditional group, was unique in that he did not complete any of the problems. This does not mean he was less capable than the other students. In fact, his comments indicate that he was able to realize the key feature of the problems that made them so difficult; namely, the non-constant force. Because of this, he was unable to rely on past practice or experience to help him attack these problems. Consider his comments at the end of problem 1:
So I could solve for the initial force due to gravity from the earth but that will of course change as the satellite moves in only, in direction and magnitude so, I'm kind of in trouble… I really don't remember doing any problems like this so far. I guess I'll stop then.

at the end of problem 2:

The ball's traveling and it's supposed to go in nice projectile motion and end over here but, it won't go as far, because there's air resistance. However, it doesn't affect the motion of the ball in a very simple manner, so I'm kind of stuck. I don't know, I guess that's it, I'm missing something. A little, a little bit harder than our homework problems.

and at the end of problem 3:

The block's gonna be swinging all around and bouncing, back and forth on the spring as it moves, in the weirdest directions, and I have no clue how I would figure out where this ends up after six seconds. I'm not sure this is possible but it seems like it should be, because I have so much information. But, seems like if it is possible, this would require some really complex equations that I don't know where to begin with, so... guess that's it for this one too.

Josh simply realized that he had never faced such complicated problems and assessed, realistically, that he would not be able to complete them. This is, in some sense, more thoughtful than other students who attempted to solve the problems through invalid means, such as using circular motion methods.

3.7 Discussion

Experiment 1 confirms the findings from the pilot study that students from traditional courses, who are not explicitly taught computational or iterative methods, are not able to invent an iterative method to solve difficult dynamics problems. While some students from the M&I course did decide to use computational or iterative methods, they did not as a group choose them overwhelmingly, even when they had a computer with the appropriate tools at their disposal. One reason the M&I students may be reluctant to use computer modeling is that they had never been given the opportunity during the course to choose between analytic or computational methods when solving a problem. The experimental sessions turned out to be the first time ever that these students had to make
such a choice. In the course, computer modeling problems were always designated as such and always assigned during designated laboratory times. All other homework problems were done on paper typically using analytic methods. Students did work exercises where they had to apply the iterative procedure by hand, but this was usually over just one or two time steps, due to the tedious nature of the calculations.

Furthermore, students in general have had years of practice solving mathematical problems analytically. Even after exposure to computer modeling in the M&I course, they likely stick to the problem solving habits that have served them in the past. Because computer modeling is new and unfamiliar, some students are uncomfortable with it, do not understand the point of it, or just find it difficult.

Revised instruction might help alleviate student apprehension to computer modeling. Students may need more opportunities to practice choosing between computation and analytic methods. Instructors could devise assignments that allow students to use either means to solve problems. Students may also need more guidance on recognizing the purpose of computer modeling and types of problems where it is useful.

Regarding the problem solving approaches, the protocols do suggest there are some qualitative differences between the M&I and traditional students. The M&I students' greater use of the momentum principle in the form \( \Delta \vec{p}_{\text{total}} = \vec{F}_{\text{net}} \Delta t \), and their greater likelihood to use it in an algorithmic way to update velocity and position, is likely due to the emphasis placed on it in instruction. The connection between Newton's laws and momentum are not often made clear in the traditional curriculum, nor are Newton's laws or momentum often used for the prediction of motion; rather, they are usually
applied to systems where the motion of a body is known or constrained. Furthermore, constant acceleration kinematics equations are often given great emphasis at the beginning of the traditional course, which may lead students to have an exaggerated sense of their importance or centrality. While the M&I students did use these equations at times, they usually were falling back on them after reaching an impasse in trying to apply fundamental principles. The M&I students in their course had practiced solving many problems where the motion of bodies were unconstrained and unknown; this may have contributed to them being less likely than the traditional students to fit certain problems to the template of circular motion. Greater exposure to such problems also seemed to give the M&I students a greater expectation that they could be solved, considering that the M&I students gave up less often than the traditional students.

It is important to point out that the M&I students were not necessarily better students or more capable problem solvers than the traditional course students, and it would be a mistake to compare the "rightness" or "wrongness" of the solutions of the two groups. Indeed, students from both groups often made similar mistakes in physics concepts and mathematical calculations (which will not be discussed here). The important difference is that the students from the two courses made use of different conceptual or problem solving tools, due to the differences in the curricula. The conceptual tools that the M&I students used, particularly the discrete form of the momentum principle, are also used in computer modeling. Therefore it might be possible, with some improved instruction, to help M&I students invoke these conceptual tools in the context of computer modeling. The tools that the traditional course students relied on ($F=ma$, kinematics equations, etc.) are not as appropriate for computer
modeling, so it would require a greater amount of explicit instruction to teach these students computer modeling methods.
4 Experiment 2: The VPython programming study

As discussed in section 1.4, the results of experiment 1 opened up opportunities for further research in a number of areas related to $M&I$, including students' abilities in problem solving and computer modeling. I chose to look in more detail at the specifics of computer modeling in experiment 2, which was conducted to address the following research questions (questions 3 and 4 from section 1.4):

- What difficulties do students in the $M&I$ course have with computer modeling?
- Can tutorial-style instructional interventions help alleviate these difficulties?

Experiment 2 took place in two phases. The first phase of the experiment was conducted in Spring 2003 at North Carolina State University. Using the data gathered in phase 1, instructional documents and activities were designed and used at the beginning of an $M&I$ based introductory physics course at NCSU in the Fall 2003 semester. The second phase of the study was then run, following the same procedure as the first, to observe any differences between the two courses, and to see if these changes could be attributed to the redesigned instruction.

4.1 Participants

In phase 1, the participants were four paid volunteers from the Spring 2003 $Matter & Interactions$-based introductory mechanics course for engineering students at NCSU. This course was taught in the SCALE-UP environment developed by Robert Beichner and colleagues at North Carolina State University (Beichner et al., 2000). SCALE-UP is a highly interactive studio classroom environment for large classes. Mini-lectures, group problem solving, laboratory activities, and VPython programming all took
place in this one classroom setting. The participants in phase 2 were five paid volunteers from the Fall 2003 Matter & Interactions-based course for engineering students at NCSU. The course was taught in a traditional lecture setting with weekly laboratory sections. In the laboratory sections, students were introduced to VPython programming and worked on programming assignments. The relationship between physics and programming was at times discussed by the instructor during the lectures.

Table 4.1 lists the pseudonyms of the students that participated and their final grades in the course. All the volunteers were male students. It is unfortunately the case that, on average, the volunteers for the first semester did not perform as well in the course as the volunteers for the second semester, which makes direct comparison difficult. The low grades of the students in the first semester were not entirely due to programming. Norman, in particular, disengaged from the course in the latter half of the semester, even missing exams, but his performance on programming tasks in the early part of the semester was acceptable. Despite the low grades, there is little reason to believe that the difficulties these students had are not representative of the problems that often occur in the Matter & Interactions course. Instructors of the course have often reported difficulties occurring in class programming activities that are similar to the ones I will discuss in this chapter.

Table 4.1. Volunteers for Experiment 2 and the final grades they received in their introductory physics courses.

<table>
<thead>
<tr>
<th>Phase 1: Spring 2003</th>
<th>Phase 2: Fall 2003</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pseudonym</strong></td>
<td><strong>Grade</strong></td>
</tr>
<tr>
<td>Bobby</td>
<td>C</td>
</tr>
<tr>
<td>Norman</td>
<td>F</td>
</tr>
<tr>
<td>Paul</td>
<td>B</td>
</tr>
<tr>
<td>Richard</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.2 Experimental procedure

Each participant was interviewed individually and privately in three one-hour-long sessions over the course of the semester. Each interview session was separated by two or more weeks. The first session took place after the students had already been introduced to VPython programming in the course and had turned in the first programming assignment. In the first two sessions, the participant was asked to work on a computer program that was similar to one that had been assigned in the physics class. While working on the programs, participants gave think-aloud verbal protocols. Participants were instructed to do as much as they could on their own, but if they reached an impasse, they could ask me for help, or use VPython's online help pages. The online help only gives details on VPython syntax and graphical objects, not on physics.

In phase 1, the programs that students wrote in each session were the following:

- **Session 1**: Two separate tasks were assigned:
  - Program 1: Create a scene of a table and chairs of specified dimensions and at specified locations using cylinder objects, and make certain relative position vectors using arrow objects. This task was analogous to an in-class assignment, where students had to create 3-D model of the circular tables in the SCALE-UP classroom.
  - Program 2: Model the motion of an asteroid moving in free space at a constant velocity.

- **Session 2**: Model the motion of the moon around the earth. If time remains, create arrows representing the moon's momentum and the gravitational force
on the moon. This was similar to a program assigned in class to model the earth's orbit around the sun.

- Session 3: Arrange a group of randomly arranged lines of VPython code in the correct order to make a working orbit program.

In phase 2, the only task that was significantly altered was program 1 of session 1. Here, the participants had to create a static model of the planet Jupiter and three of its moons with sphere objects, and show various relative position vectors with arrows. This task involves using sphere and arrow VPython objects; it does not require the use of the cylinder object as in the first task of phase 1. The new task for session 1 reflected a change in instruction in the second semester that was partly based on findings from phase 1, and partly due to practical considerations due to the move from the SCALE-UP environment to the traditional lab setting. I will describe the reasons for this change in the next section. For the exact instructions and problem statements that were given to the students in the experiment, as well as examples of completed programs, see section 7.3 of the Appendix.

4.3 Instructional approaches

A variety of changes in the instruction of computer modeling were made in the second semester. These changes resulted from efforts to use research results from phase 1 to attempt to improve instruction, and from practical considerations due to a change of course setting. I will describe the instructional approaches that were used in the two semesters to introduce the syntax and semantics of VPython and to introduce computer modeling of motion.
4.3.1 Computer modeling instruction in Spring 2003

In the Spring 2003 semester, when the M&I course was taught in the SCALE-UP environment, students were introduced to VPython and computer modeling through a variety of tasks. The instructional sequence began in the first week with an activity designed to introduce students to 3-D vectors. In the non-traditional SCALE-UP classroom, small groups of students sit at round tables. The task required students to measure the x, y, and z coordinates of the center of their tables with respect to a chosen origin. Based on their collected data, they had to calculate the magnitudes of the vectors, give corresponding unit vectors, and calculate relative position vectors between selected tables. (See Appendix, section 7.4.) Next, students worked on a tutorial document that introduced them to the basics of VPython, including how to create, save, and run a program, and the basic syntax of cylinder and arrow objects. This tutorial document was not an interactive worksheet; students merely read it and followed its instructions by typing in their own programs what they were told to type. (See Appendix, section 7.5.) After this, students had to write a program that used their collected data to create a 3-D model of the classroom. Cylinder objects represented the tables, and arrow objects were used to show relative positions between certain tables. (See Appendix, section 7.6.)

In the next class meeting, students were introduced to the concept of velocity and how to use it to calculate changes in position over a short time. The instructor then introduced the students to using the velocity concept to change the position of an object with VPython. The instruction was a "follow-along" sequence, where the instructor typed each line of code on a screen that was projected on the wall for students to see, so that they could type the same lines into their programs. The instructor first showed how a
single application of the position update line would make a ball move one step, and then introduced loops to show how to continually update position. In this program, the object moved at a constant velocity. One week later, after students had been introduced to the momentum principle, students revisited this program. In the same sort of "follow-along" sequence, the instructor showed students how to modify the program to model the motion of the object with a force exerted on it. (See Appendix, section 7.7).

4.3.2 Computer modeling instruction in Fall 2003

Between phase 1 and phase 2 of the experiment, instructional materials were developed and used at the beginning of the Fall 2003 semester in the labs of the M&I course. These materials were tutorial style documents that guided students through the initial stages of learning VPython. They were based on some of the preliminary findings of phase 1. The research purpose of these materials was to see if a more carefully guided introduction to VPython, one that more clearly focused on some of the early stumbling blocks, could help alleviate later difficulties in programming. These materials were also necessitated by the fact that the M&I course moved from the SCALE-UP environment to the traditional lecture and lab setting. Programming in the NCSU M&I course was now being introduced in the laboratory setting by teaching assistants who were inexperienced with VPython and with the curriculum. A modular, portable way for multiple TA's to quickly introduce their students to programming was now needed.

The change of venue affected the development of the instructional materials. In the first semester, students in the SCALE-UP environment used cylinder objects to model the circular tables of the classroom in the first programming assignment. In the second semester, students in the laboratory settings were not in this special classroom, so there
was little reason to introduce cylinder objects early in the course. (They were, however, 
eventually taught how to create cylinder objects later in the course, in the context of a 
model of mass on a spring.) Furthermore, it was found in phase 1 that students had 
Difficulties constructing cylinders. As a result, the new instructional materials do not 
introduce the creation of cylinder objects; considering that one of the goals was to help 
students become familiar with relative position vectors in three dimensions, cylinder 
objects were deemed superfluous at this point. Consequently, the aforementioned change 
in the first programming task for phase 2 of the study (modeling a room with cylinders 
changed to modeling a planet and static moons with spheres) was made to reflect what 
was taught.

The instructional materials consisted of several parts that were used in the first 
two lab sections of the semester. The first part was a worksheet whereby students did 
exercises and calculations that reviewed vector concepts, including components of 3-D 
vectors, addition and subtraction of vectors, scalar multiplication, and unit vectors (see 
Appendix section 7.8). Many of the exercises in the worksheet focus on drawing and 
analyzing vectors whose tails are not at the origin. This serves as preparation for dealing 
with a confusion that students in phase 1 had between the location of an arrow object's 
tail and the vector components of its axis in VPython.

The second part was a tutorial introduction to VPython programming (see 
Appendix section 7.9). It first introduces students to the editing environment, the 3-D 
display, and the text feedback window that provides error messages and text output. 
Next, it introduces how to create sphere objects and arrow objects, and the purpose of 
their various attributes, such as color and position. Special attention is paid to arrow
objects because of their importance in representing vector quantities. The tutorial highlights the difference between the "pos" attribute of an arrow, which is the vector that gives the position of the arrow's tail with respect to the origin, and the "axis" attribute, which is the vector components of the arrow with respect to its tail. Students in phase 1 of the study often confused these attributes. The tutorial asks students to explore the difference between the two attributes by making arrows that have the same "axis" but different "pos", giving parallel vectors based at different locations. Relative position between two objects in space is introduced as the vector subtraction between their positions (final position minus initial position). Students calculate on paper appropriate relative position vectors and then use them to construct the corresponding arrow in the 3-D display. Scaling an arrow's axis through scalar multiplication is also explored. Next, the tutorial introduces how to refer to an object's attributes symbolically so that it can be used in general formulas; for example, once defined, the "pos" of an arrow named "arr" can then be referred to as "arr.pos". Students do paper-based and computer-based exercises with this notation. The last section of this tutorial introduces the basic concept of a loop. Here, students create a loop that has only one calculation in it—a variable is incremented 10 times in a row, printing the result after each addition.

In the following lab, students are introduced to modeling motion with a computer in the context of a motion cart experiment. Students first work on a lab experiment in which they analyze the motion of a cart that is free to move on a linear track with minimal friction. Students analyze two cases: first, students push the cart so that it rolls with a constant velocity, and second, a constant force is exerted on the cart by a motorized fan. Next, a tutorial guide introduces to the students how to simulate this
experiment with a computer model (see Appendix section 7.10). The tutorial starts by emphasizing that all computer models have two major parts: pre-loop set-up, and a dynamics loop. After instructing the student to create objects that represent the cart and track, the tutorial introduces the concepts of initial conditions (in particular, initial momentum) and time step. Next, the "while" loop is re-introduced, and the tutorial shows in detail the calculations needed to update the position of the cart when it is moving at a constant velocity. Following this, the momentum principle is introduced, and it is linked to the method for modeling the motion of the cart under a constant force. Special attention is paid to mapping the algebraic notation to the VPython syntax (see figure 4.1).

Figure 4.1. Diagram from the instructional materials designed and used for the second semester. It is one of example of explicit mapping of algebraic notation to VPython syntax.

The tutorial worksheets in the second semester were more interactive than the instructional documents used in the first semester, and they were better suited for self-paced work by the students. Unlike the first semester, students in the second semester had to calculate and make predictions of what should go into a program before actually
typing into the program (for example, the values of vector components, or symbolic expressions in VPython syntax for relative position vectors). Students were told at certain points in the worksheets not to go on to typing until an instructor had checked their written work. In the first semester, students at first followed all instructions given by a document or by the instructor, but later had to do a major task on their own; there was no intermediate stage of coaching. The new materials also tried to elucidate problematic concepts that were seen in phase 1 of the protocol study. More practice was given on vector concepts, and efforts were made to connect paper-based vector calculations to similar calculations with the computer. Special attention was paid to confusion between "pos" and "axis" attributes when introducing arrows. When introducing modeling motion, the new tutorials specifically focused on showing in detail the overall structure of a program.

4.4 Nature of the interviews

The programming tasks that students worked on during the experimental sessions were extremely complex. Each program consists of a number of multi-step procedures that require competent knowledge of a large number of physical and mathematical concepts, such as force, the momentum principle, velocity and position, vectors and vector operations, in addition to the syntax issue of the programming language. Had students been left alone to work without interviewer intervention, as is done in a classic think-aloud protocol study, it was clear that most of them would have quickly gotten to a point where either they would not know what to do next and then simply stop working, or they would become overwhelmed and hopelessly confused. This was undesirable for two
reasons. First, it would be unethical to let students flounder on such complex problems for any sizeable amount of time. Remember that students had already successfully written similar programs for class assignments, but there was much more help available to them for their class work. Programs were typically started in class or lab, with instructors at hand to answer questions. Students also typically worked in groups of two on programming assignments, and could refer to previous work as examples. These resources were absent in the experimental setting. Students who believed they were fairly competent at computer modeling could become greatly discouraged by their performance in the experiment if not offered any assistance. Since these students were enrolled in M&I class while they were participating in the study, any discouragement resulting from their participation in the study could have adversely affected their performance in the class.

Second, letting students reach an impasse without intervening would not fully address the research question. When solving a problem by hand on paper, a student could make a mistake on one part of the problem, but still go on to correctly invoke concepts or make calculations in later parts of the problem. This is difficult, and sometimes impossible, to do with a computer program. Each line of a program must be syntactically and logically correct for the program to run and produce the desired output. Often when a student made a mistake on part of the program, the program would not run; it would instead produce an error message. If the student was unable to correct this mistake, he would be unable to proceed with the rest of the program. Therefore, we might learn that the entire process of computer modeling is difficult, but it would shed no light on which pieces of the overall procedure students find difficult, and which they are
able to understand. For example, many students in this study had trouble with the initial stages of the program, where graphical objects have to be created. Without help in correcting this early part of the program, students could not move on, and there would be no opportunity to observe their understanding of parts of the program more central to physics concepts, such as the force calculation or momentum principle. Therefore, it was at times necessary to help students past a piece of the program that they found difficult in order to move on to the next piece.

For these reasons, the dialogue in these verbal protocol sessions often diverted from a traditional think-aloud session to something closer to a tutoring session or a demonstration interview (see section 2.3.3 and 2.3.4). In making the decision to intervene, I used the following general guidelines.

First, I intervened whenever students made a mistake in spelling or punctuation. Many students would often forget equals signs or parentheses when writing lines of code, such as the ones needed to create VPython objects, or misspell words. These errors were usually flagged when the program was run. (I encouraged students to run the program frequently to test for errors.) If students could not immediately see the mistake, I would ask them if they understood the error message. If not, I would simply tell them to make the necessary correction. Since I was more concerned with students' understanding of the physics concepts in the program, and not the specifics of VPython syntax, I had no qualms about offering help on these issues.

Similarly, sometimes students did not remember VPython-specific keywords. For example, many students remembered that they needed to create 3-D objects, but could not remember the VPython keywords needed to do so. I would then tell them the
keywords needed to define the objects, such as "sphere", "cylinder", or "arrow". I would NOT tell them the particular attributes at first though, such as "pos" or "axis", because these referred to geometric and physical information that they needed to define.

For any lines that dealt with physics or geometry (the majority of the program), I let students proceed without intervening until they reached an impasse. In order for me to intervene, the student would have to ask me for help or a hint on a concept or a line in the program, or state that he could not go on. At such a point, I would offer some interventions, usually in the form of leading questions. My approach was at first to offer very general probes, such as "What do you think the problem is?" or "Why is this behavior occurring?" in order to keep my influence on student thinking at a minimum. If students still struggled, I would then offer more specialized probes that referred to the specific difficulty. Once the student successfully completed the particular part of the program he was struggling with, I would again "back off" and allow the student to proceed unaided.
S: So moon dot p equals...

[Student types in program] `moon.p=moon.p+Fnet*deltat`

S: Ok. So moon dot position is gonna be...

[Student types] `moon.pos=`

I: I'll get you another sheet of paper if you want.

S: Ok, it's gonna be, let's try again, moon.p, no, delta p equals...

S: Hm...

[Student pauses for a few seconds]

S: Ok. He asked us the same, in the very first or second quiz...

I: That's right.

S: That's the only thing that's missing in here, I'm sure of it. Ok how would I do it, the delta...

[Student pauses for a few seconds]

I: What are you trying to do, I guess, what are you...

S: I remember what I did during that time was use some relationship between p and mass and velocity, p equals m v.

I: Ok.

S: I used that somehow to get the formula.

I: Ok. Well, ok. So, so you need, so momentum is gonna come into it somehow.

S: Mm-hm.

I: This line that you're trying to write right now, what's the purpose of that line? It does what?

S: It updates the position of the moon, so that's why it rotates.

I: Ok. So it's gonna change the position of the moon.

S: Mm-hm.

I: So, what do we need to know?

S: It should be moon dot p o s, plus...ok...

[Student edits a previously typed line to read:] `moon.pos=moon.pos+`

[Student looks at his written work again]

S: Ok that's r minus r 1, that's that...plus somehow I need to use that, so it's gonna be...ok it's gonna be moon.p times delta, delta t...over moon dot m

[Student finishes editing this line, and it reads:] `moon.pos=moon.pos+(moon.p*deltat/moon.m)`

**Figure 4.2.** Example of dialogue from session 2, moon orbit program.
Figure 4.2 is an example of a dialogue transcript. It shows the general pattern of interviewer questioning: intervene when the student reaches an impasse, and gradually shift the questions from more general and less obtrusive to more specific. In this excerpt from the moon orbit programming task in session 2, a student has reached the point in the program where he has typed the momentum principle, and now is trying to decide how to update the position. Because he had previously written some work on paper, I offer him a new sheet of paper. After thinking about it, and realizing it is the only step missing from his program, he believes he is stuck and asks for a hint ("how would I do it, the delta..."). At first I ask a general question to get him to explain what he is thinking. He volunteers that momentum is involved, but he apparently does not know the correct relationship yet. I confirm his idea that momentum is involved, and ask a more specific question on the purpose of the line he is writing. He correctly states that it is to update the position, and after I again confirm this, this seems to be enough for him to finally recall the correct formula on his own.

4.5 Qualitative discussion of student difficulties

Transcribed verbal protocol data from the sessions were analyzed both qualitatively and quantitatively. The purpose of the qualitative analysis was to explore in detail the difficulties students had while writing the computer models. The quantitative analysis more rigorously measured the amount of student difficulty in order to compare the results of the two phases of the experiment and to judge the effectiveness of the revised instruction. I will describe the quantitative analysis methods in section 4.6.

The qualitative discussion of student difficulties in this section will be limited to
the moon orbit program from session 2. The data from session 2 are of particular interest for several reasons. First, the moon orbit program is most representative of the types of programs that students write in the *Matter & Interactions* course, because it encapsulates the physical modeling algorithm (initial conditions, force calculation, updating momentum and position, and repeating) emphasized by the course. The program mixes physical, mathematical, and geometric concepts, unlike program 1 in session 1, which primarily dealt with 3-D vector geometry. If a student could successfully write the moon orbit program, it would be a strong indicator that he understands the most important physics concepts emphasized by *Matter & Interactions*. The second is a practical reason: this program was the only one common to both phases that all students wrote entirely from scratch. As mentioned, program 1 in session 1 differed between the two phases. Not all students completed program 2 in session 1, due to lack of time in the session. The program in session 3 was not written from scratch. So, session 2 best allows for comparison between the two phases on the complex task of writing a physical modeling program from scratch.

This discussion will focus on the difficulties with the major steps necessary to write a computer model of an orbit program, such as declaring initial velocities and momenta, defining a time step, and creating a physics loop that includes calculations for force, momentum, and position. These steps will also be used as categories for coding in the quantitative verbal analysis (see section 4.6). This analysis is intended to reveal difficulties with physics and mathematics concepts as they show up in the context of programming. Syntax mistakes (such as spelling errors, forgetting commands to create graphical objects, placement of colons or parentheses) are less important to the
instructional goals of the *Matter and Interactions* curriculum, and I will not talk about them in detail. (Syntax errors were classified and counted in the quantitative analysis.)

4.5.1 *Program purpose and structure*

One of the main goals of the *Matter and Interactions* course is to teach students the basic structure of a computer model: initial conditions and constants are first defined, then a loop iteratively updates the dynamic and kinematic quantities that change with time. In knowing this, it is important for students to understand that a computer executes a program line by line, sequentially. Before a quantity can be updated, it must first be defined or initialized in a line that comes earlier in the program. Also, merely writing a law or principle in the program once does not cause the computer to "remember" when to execute it; the law must be placed in the proper sequence—a law or principle used to continually update a quantity must occur in the loop. These ideas are related to Bruce Sherin's symbolic forms of SEQUENTIAL PROCESSES, SETUP-LOOP, and TRACING (see the discussion in section 2.1.3). Two students from phase 1, Bobby and Richard, and one from phase 2, Andrew, had difficulties with the structure or purpose of the computer model; they each had interventions that were coded in the "Loops and program structure" category.

Bobby's approach in session 2 illustrates a misunderstanding about the order of lines and the purpose of the loop. After creating the earth and the moon objects, he defined the time step and time, and wrote the "while" statement, saying "I guess I'll try the while loop now." After this, his next step was to "set the force and momentum for the moon," and he defined an initial momentum. He then worked on writing an expression for the force, which he wrote *before* the while loop line (see figure 4.3).
At this point he stated that he is "not quite sure what to do next." I began to intervene, and in order to see if he could realize his structural mistakes on his own, I first concentrated on helping him correct his expression for the force. After working through this, I asked him for his next step:

I: Ok. So now you have, let's stop and go back and see what you did. You now have all the pieces that you need to calculate a gravitational force. We're still not done yet, but we have one big, one big chunk of it, of this program, completed. So what would be the next thing we have to worry about?

S: The momentum principle?

I: Ok, let's talk about the momentum principle. Where's that going to come, how're we gonna write that, et cetera?

S: It would come after the force.

He knew that the momentum principle comes after calculating the force, but he again wrote it before the loop, not realizing that both the force and momentum calculations go into the loop. But after calculating the momentum, he next wrote a line to update the position, which he put in the while loop:

I: Once you've, once the thing has a new momentum, what do we use the momentum to do?
S: To update the position?
I: Yes.
S: So it would go into the while loop now?
I: Yeah, let's put it in the while loop.

Once he had the position update line, he had all the major pieces of the program, but not in the correct order, so I decided to ask about this issue:
I: Let's talk about the structure. Um, you've got right now all the pieces. Everything we need to make this run is in there right now, it's just a question of the ordering of the program. Um, why do we need a while loop? What is the purpose of a while loop? What does that do?

S: Um, to repeat certain parts of the code, of the time period.
I: Okay, what things do we need to repeat?
S: Uh, need to...need to find...r?
I: Yes.
S: And the force on the moon.
I: Mm-hm.
S: And the momentum.
I: Yes, yes.

So after this prompt, Bobby realized that momentum and force calculations must go in the loop, and made the necessary corrections.

Another important idea involved in computer modeling is that behavior of the modeled physical system is not pre-determined. That is, the trajectories of the particles are not specified by solutions to the differential equations (the momentum principle and the position update equation, written in discrete form); rather, the trajectories emerge from the application of differential equations themselves, seeded with initial conditions. Different initial conditions may result in different trajectories, but the force law, momentum principle, and position update equation remain the same. Richard seemed to have difficulty with this concept. He believed that to make the moon orbit the earth, he needed a specific equation or formula for a circular orbit. For example, after he created the graphical objects, he stated that he wants to make the moon orbit the earth once in 28 days, and said:

S: And, I would do that by, wondering what, formula, or I need to set the moon to have a certain, uh, velocity first. So, the velocity would be, we need to label that, I guess.

[He types:] velocity.moon =

S: ...Dot moon equals...um....I would need to find the radius....thinking if I used, or if I found the radius, then I could go ahead and use the circular orbit equation.
At this point we talked about using the radius to calculate an initial speed, and after some prompting, he came up with an expression for the initial speed. He then came back to the issue of orbital motion, and said:

S: Ok, and...now that I know how fast it needs to go, need to figure out, um, how to make that velocity go in a circular orbit. So I'd use the...um...m omega squared r, so I'll say...moon dot orbit, I guess?

S: Um...trying to think about how I can tell it to make it go in that kind of an orbit.

The formula that he mentioned is actually the equation of motion for circular motion,\[|\frac{d\mathbf{p}}{dt}| = m\omega^2 r,\]where \(|\frac{d\mathbf{p}}{dt}|\) is the magnitude of the time derivative of momentum, \(m\) is the mass, \(r\) is the radial distance, and \(\omega\) is the angular speed. Although it would be possible to write a computer model of circular motion by applying this equation, it would not be a general, flexible model, because the motion would always be constrained to a circle, regardless of initial conditions. Even if a model was written to constrain motion to a circle, the equation would need to go into a loop. Richard never mentioned the need for a loop.

I then tried to see if Richard could recall the procedure he used for a different program from class, in which he had to model the oscillatory motion of a mass on a spring:

I: Ok, let's think about a, another program. You did the spring program. Right? Um, what kind of path did the ball, or the block at the end of the spring take? What was its motion?

S: It was oscillating.

I: It just, ok it just oscillated back and forth. How did we make it do that?

S: Um, let me think. I set the velocity to go between a certain interval, I think.

But this is incorrect. The changing velocity was never written into the program. The initial velocity is defined only once, and the force law and momentum principle leads to the oscillation.
At this point it was clear that Richard was struggling with the basics of creating a model, so we moved away from this topic. Instead, I guided him in how to model simple constant velocity motion. We talked about the need for a loop, and how to update position with a loop. But creating a constant velocity model did not seem to jog his memory. When I asked him what other equations or principles might go in the loop, he again responded, "Circular orbit."

Andrew from phase 2 also had a confusion similar to Richard's, but it was not as deeply ingrained. Andrew, like Richard, brought up "omega" when calculating an initial speed. Since angular speed $\omega =$ $\frac{2\pi}{T}$, where $T$ is the period, he first calculated $\omega$ (by hand), and then calculated the initial speed $v = \omega r$, the result of which he defined in the program. When he came to the while loop, he knew he first needed to find "rmag", the distance between the earth and the moon, but asked for help on what came next. I tried to prompt him to remember how rmag is used in the next steps of the program:

I: Ok. Well first of all, what's it refer to? What is rmag? What is the thing that, what is it defining, or what is it calculating?
S: Um, well it's calculating the magnitude of the vector between the earth and the moon.
I: Ok. It's the magnitude of the vector between the earth and the moon. Um...why is that vector between the earth and the moon important? What do we need to use it for?
S: To be able to calculate the speed.
I: Speed. Ok, um...how is the speed calculated?
S: By this um, omega r.

I explained that while it is possible to use this method to model the motion, it would result in only circular orbits, and would not be generally applicable. I then asked him about the purpose of the loop:

I: What are we, what are we doing inside a while loop? What's going on in a while loop, why do we need a while loop?
S: To tell it to like change positions.
I: Ok, so things that are going in the while loop are things that are going to change.
S: Right.
I: Ok, and one of those things is position. What else is changing?
S: Momentum.
I: Ok, so the other thing is momentum. So we're gonna need a way to tell the computer in the loop to change the momentum by some appropriate amount...
S: Mm-hm.
I: ...and to change the position by some appropriate amount.
S: Right.
I: Ok? So...what do we need to do in order for it, in order for it to change the momentum?
S: Um...change in momentum equal to the net force times the change in time.

So unlike Richard, some prompting enabled Andrew to abandon his pre-determined circular motion approach and use the momentum principle.

4.5.2 Time and time step issues

In a computer model of motion, the time step (referred to algebraically as \(\Delta t\) or in VPython code as "deltat") is the unit of simulated time over which a modeled object's new velocity, momentum, and position is calculated for every iteration of the physics loop. A running counter of simulated time can be kept by including a line of code in the loop that increments the simulated time by one unit of the time step in every iteration. The running time can be used as a condition for the loop's termination. That is, the "while" line states that the loop be executed while the value of time is less than some final stopping time. Figure 4.4 shows an example of how this is written in a VPython program. First, before the loop, the time step \(\text{deltat}\) is defined. Then the simulated time, called \(t\), is initialized, or set to zero, much like resetting a stopwatch. If we consider that \(t\) represents time in units of seconds, the "while" line tells the computer to keep executing the loop while \(t\) is less than 28 days. In fact, at the beginning of each iteration of the loop, the condition "\(t<28\times24\times60\times60\)" is evaluated. If it is true, the loop is
executed; if not, the loop is skipped and the program continues. After the physics statements in the loop, the value of $t$ is increased by $\text{deltat}$, and the loop repeats.

```python
deltat = 100
$\text{t} = 0$
while $\text{t}<28*24*60*60$
    # Physics goes in loop...
    $\text{t} = \text{t} + \text{deltat}$
```

Figure 4.4. Using time and time step in a VPython computer model.

Students in both phases had difficulty with the concepts of time and time step. Only Kyle from session 2 did not have any interventions on this topic. A common problem was a belief that the time step, $\text{deltat}$, should be 28 days, which is the period of the moon's revolution around the earth. Bobby, Paul, Norman, and Andrew all made this mistake. The source of this confusion could come from a variety of interfering ideas about the concepts of period and change in time.

Andrew, for example, knew that he needed to define a quantity called "$\text{deltat}$," but he seemed to believe that it indicated the total time over which the program runs. He did not, at first, mention any relationship between $\text{deltat}$ and the physics calculations in the loop. Consider this excerpt from his session 2 interview:

S: Ok. Alright so then, I gotta, I have a deltat right?
I: Ok.
S: Ah, equals, now what is this, what is this, change in time supposed to, is it supposed to be the 28 days, or...?
I: What is, well ok, let's talk about, what is the meaning of deltat? What does that mean, what is it used for?
S: I mean, it's used for, it means the change in time.
I: Ok. Um...it's change in time. And in particular, what do we do with it? In the, like in the rest of the program, how is it used in the rest of the program?
S: Um...
Andrew paused for a few seconds, and then continued:

S: Isn't it like, for how long the thing, the program is supposed to run for?

So he believed at this point that \( \text{deltat} \) indicated the total running time of the program. The total running time does in fact need to be defined, but this occurs in the "while" statement.

Since he did not yet know what \( \text{deltat} \) should be, we decided to move on. The next step that he dealt with is creating the loop, and the issue of time came up again. After typing "while(t<" in the next line, Andrew said, "I have to have \( t \) less than something in here...but I don't know what it is." Writing the while line apparently caused him to remember that he also needed a definition of \( t \). He said, "And I know I have to define \( t \) up here too," and types "\( t = \)" above the while line. At this point, I tried to get him to confront his confusion about time issues:

I: Ok, so now let's talk about, we have \( t \) and we have \( \text{deltat} \).
S: Yeah.
I: Ok. Um...
S: So then, \( t \) is gonna be the total time.
I: \( t \) is the total time, that's right.

Unfortunately, my affirmation that \( t \) is the "total time" may have further misled Andrew. By total time, I meant "total elapsed time", while it appears that Andrew meant "final stopping time". We continued:

S: Right. So would that be in seconds in this program, or...?
I: Yes.
S: Do you want it to run for 28 days?
I: Yeah.
S: Ok. 2419200, ok.
[Andrew types:] \( t = 2419200 \)

So Andrew now made the same mistake with regard to defining \( t \) as he previously did in defining \( \text{deltat} \): that it should be the final time to end the program. But he still seemed
to realize that the defined time cannot be the same value as in the "while" line; he knew that \( t \) must be less than some larger number in order for the program to run, for he then stated:

S: So then...while \( t \) is less then...well, so if we want it to run for...the total days, then we just put a value in that's bigger than \( t \), right?

At this point I decided to intervene mildly.

I: Ok, uh, let's think about this then. You want, you want it to run, until, we want it to run for 28 days, we want it to run for that amount of seconds.
S: Right.
I: Does that mean we should start off with that amount of seconds?
S: Ah...so if we want to start off from the beginning, and then you want to end up....
[Andrew edits the "\( t = 2419200\)" line before the loop to read:] \( t = 0 \)
S: We'll say 2419201, make sure it runs the whole thing.
[He then completes the "while" line to read:] \( \text{while}(t<2419201) \)

So it appears that by focusing his attention on the value of time that "we should start off with," Andrew was able to distinguish between time initialization and final time in while line.

The time step was not dealt with until later, when Andrew wrote an expression for updating the momentum that included \( \text{deltat} \). He then focused on the definition of \( \text{deltat} \) again, and stated, correctly, that "it's basically the step of each time it goes through the loop." But he was still uncertain on choosing an appropriate value for \( \text{deltat} \), saying:

S: And it's gonna, so I mean, do you want me to put any particular, if I just did one day, would that be fine?

In this case, one day is a reasonable first choice for \( \text{deltat} \), since the total orbital period is 28 days—the orbital motion will be broken into 28 discrete steps, corresponding to 28 iterations of the loop. Students in the *Matter and Interactions* course are taught that the value of \( \text{deltat} \) should be small enough so that the force is approximately constant over
the interval, but not so small that it takes an inordinate amount of time for the computer to plot the motion of the system. The size of $\text{deltat}$ depends on the system being modeled; for orbital motion, students are taught that a trajectory that stays closed (no precession or spiraling) over several orbital periods is a good indicator that $\text{deltat}$ is small enough to produce an accurate model. It is not clear however, if Andrew takes any of this into account; his choice of "one day" could be just a guess, or a default choice for $\text{deltat}$, regardless of system.

Not all of the students had the same types of confusions between time and time step. Norman, for example, knew that time had to be less than a number greater than the period for the moon to orbit at least once, so he wrote "$\text{while } t<4320000\text{", or 50 days in seconds. He did not at first initialize time or define $\text{deltat}$. After running the program and seeing an error which said that $t$ was not defined, he realized that he had to define $t$ to be equal to zero initially. After writing the momentum update statement in the loop, he then realized he needed to define $\text{deltat}$, and set it equal to 28 days. So unlike Andrew, Norman did not appear to at first believe that $\text{deltat}$ served the role of indicating the time when the program stops. But he did seem to have a poor understanding of the role of $\text{deltat}$—that it should break the motion into small steps, over which the force and momentum can be considered approximately constant, compared to the overall trajectory of the object being modeled. If the period is 28 days, then 28 days is an extremely poor choice for the time step; the force on the moon and the momentum of the moon clearly can not be considered constant over the 28 days.

It is possible that the problem itself caused needless confusion by giving the moon's orbital period. The period was given primarily as a way to calculate an initial
speed (knowing that the orbit is circular, the initial speed is the circumference/period). It would be interesting to see if students have the same misunderstandings of time and time step in a problem where period is not given, but a specific initial velocity is given instead.

4.5.3 Initial momentum and velocity issues

To model the motion of an object, it must be given an initial momentum. While the force law, momentum principle, and position update calculations in the loop are the same for a given physical system, the initial conditions (initial momentum and position) can be changed, leading to different trajectories for the objects. For the moon orbit program in session 2, the moon was defined to be in circular orbit with a known period. This gives a way to calculate what the initial momentum of the moon should be, based on its expected behavior. The student is instructed to start the moon on the x-axis (given the earth at the origin); knowing this, the initial momentum should then be in the y-direction (or negative y-direction), and the magnitude can be found by multiplying the moon's mass by its initial speed. The initial speed should be the circumference of the circular orbit (given by $2\pi$ times the known orbital radius) divided by the period of the orbit. If initial velocity is defined first, the initial momentum must be declared using the definition of momentum (for speeds low compared to that of light), which is the mass times the velocity.

Several students had difficulties with defining initial momenta or velocities. A common problem was a lack of consideration that momentum or velocity should be a vector quantity. This was a recurring difficulty that affected several students' approach to the whole program; see especially section 3.5.4 on difficulties with the force calculation. At other times, when students did realize that the initial momentum was a vector, it was
often defined to be in the x-direction instead of the y-direction; because the moon's initial position was on the x-axis, this would send the moon directly away from the earth. Some students even believed, at least as a first guess, that the initial momentum should be zero, which would cause the moon to fall into the earth. These errors in direction might reflect a lack of understanding of the physical meaning of initial momentum. Another problem that occurred, perhaps more with geometry and algebra than with physics, was that some students did not know or remember how to calculate an initial speed based on the circular orbit \( \frac{2\pi r}{T} \). Some students had difficulty remembering the definition of momentum, or had confusions between the momentum definition and the momentum principle, when attempting to find momentum given a velocity, even though these concepts were greatly emphasized in the course.

Bobby, for example, knew that he needs an initial momentum for the moon, but does not know what it should be. He asked me, "Can I just use any momentum?" I decided that it would be better for him to proceed with the parts of the program he knew how to do, so I told him to use any value for momentum, which we could fix later. He decides to define the momentum vector to be "moon.p=vector(10,0,0)". Because of the large mass of the moon, this momentum (in SI units) is negligibly small; also, it is in the wrong direction. Later, after Bobby had a working program with correct physics, he ran the program and saw that the moon fell directly into the earth. I then discussed this with him:

I: Ok, it flew right into the earth, which means what?
S: Uh, it's uh, there's not a momentum in the, direction, orbiting it.
I: Right, what direction do we want that momentum to be in?
S: Uh, the y direction?
I: Ok. So where do we fix that?
[Several second pause.]
I: What's the momentum at the very beginning of the program before we start doing anything else?
S: Oh.
[Bobby edits the initial momentum:] \texttt{moon.p=vector(0,10,0)}

Bobby corrected the direction of the moon's momentum, but because it still had a negligible magnitude, the program behaved the same way when it was run again. I intervened on this point:

I: How, let's see, um...right now the momentum is zero, ten, zero. The moon's mass is 7.4 10 to the 22. What, uh, what then is the initial speed of the moon.
S: There would be hardly any momentum in the...

At point, Bobby realized his momentum was too small, and he began to try random values. A momentum magnitude of "10e5" was too small, while "10e50" was too large. Apparently by accident, his next guess of "10e25" was about the right order of magnitude, since it led to a hyperbolic trajectory. After some more refinement, he found that "5e25" led to an elliptical orbit. He then asked if there was a better way:

S: Is there a way to calculate the...
I: Ok, so we can play around with it, or we could just do a calculation. Um, if something is in a circular orbit, could we find it's speed? Knowing, knowing the amount of time it takes to go around the orbit once?
S: Yeah.
I: Do you remember how to do that? You can write it out if you want.

Bobby was able to remember how to calculate an initial speed at this point, and he wrote calculations that defined the circumference and the initial speed. But he then got stuck on how to use this initial speed to get an initial momentum. I then intervened on this point:

I: How are momentum and velocity related?
S: Hm...
[A long pause.]
I: You know the definition of momentum?
S: Momentum is...net force times...time?
I: That's the momentum principle. The change in momentum is the force times the change in time. We define the momentum in terms of the velocity.
S: Can't remember what it was.
I: Uh, something times velocity?
S: Uh, the mass times velocity?
I: Uh-huh, yeah.

So Bobby apparently confused the momentum principle and the momentum definition.

Norman knew that he needed an initial velocity, and he calculated it correctly, but he defined it as a scalar. Though he knew that he needed to also define initial momentum, but like Bobby, he could not remember the relationship; I had to tell him that momentum was mass times velocity. His defined momentum was also a scalar (the product of the initial speed and the mass), and he gave no indication about whether or not these quantities should vectors. I did not broach the subject at that point, wanting to see if he could confront the problem later on his own. Later, after the computer gave an error due to adding a scalar quantity (momentum) and a vector quantity (force times \( \Delta t \)) in his momentum update line, I brought up the original problem again. I pointed out to him that his initial momentum was just a scalar, and asked him how to make it into a vector. He responded:

S: Put that as the um...as the x direction? And then just zero zero for y and z?

After I drew a picture of the earth and moon's initial position, and asked him to draw the initial momentum vector, he then saw that it should be in the y-direction.

Kyle also showed an interesting conceptual confusion between the momentum definition and the momentum principle, as well as a lack of attention to vectors. Early in the program, he correctly stated that the "initial momentum is the mass times the initial velocity, and velocity is in meter," but said he was "not sure how to get meters, though." A short time later he decided to write out "all the principles" on paper, including the
gravitational force law and the momentum principle. But then he tried to solve for the initial momentum—he wrote on paper:

\[ p_i = \frac{-GMm}{r^2} + p_f \]

while saying:

S: So initial momentum would equal negative G m m r squared, delta t, plus earth...?

When I asked if he was indeed trying to solve for the initial momentum, he backed off from this approach and said:

S: Would it be easier to do like this, say the initial momentum is equal to mass times velocity...
I: Right.
S:....where the velocity is equal to...would it position over seconds? It wouldn't be position over seconds, it would be...change in position over change in time.

He still was confused on exactly what "change in position" to use for a velocity calculation. I then hinted that circular orbits were special:

I: You, you're on the right track. There's something special about circular orbits that we know in terms of the speed. If the moon is going around in a circular orbit...
S: It's constant?
I: It's a constant speed. Ok? So you're right in saying that it's change in position over change in time...how, when you think about the speed, how much distance does it cover when it goes around one circular orbit?
[Several second pause.]
I: Ok, so...
S: Uh...do I need to put that, the, um, the equation for the circumference of a circle?
I: Ok, so you need the equation for the circumference of a circle.

At this point Kyle was able to calculate initial speed and momentum, but they were both scalar quantities. (See the next section for more of Kyle's problems with vectors.)

4.5.4 Force issues

Perhaps the most difficult part of the computer modeling procedure is to successfully construct the force vector calculation. (The quantitative analysis will show
that force caused more difficulty to students on average than any other part of the program; see section 4.8.2.) Part of the reason may be that the gravitational force law is a vector expression which combines a number of important physical quantities. One way to express the force on object 1 by object 2 algebraically is

\[ \mathbf{F}_{12} = \left( \frac{G m_1 m_2}{|\mathbf{r}|^2} \right) \hat{\mathbf{r}}, \]

where \( G \) is the gravitational constants, \( m_1 \) and \( m_2 \) are the masses of the objects, \( \mathbf{r} \) is the relative position vector that points from object 1 to object 2, and \( \hat{\mathbf{r}} \) is the unit vector that points in the same direction as \( \mathbf{r} \). In the *Matter and Interactions* course, students are taught a multi-step procedure that involves separately calculating position vector "\( \mathbf{r} \)", its magnitude "\( \mathbf{r}_{\text{mag}} \)”, and the unit vector "\( \mathbf{r}_{\text{hat}} \)." The magnitude of the position vector is then used to find the magnitude of the force "\( \mathbf{F}_{\text{mag}} \)" and finally "\( \mathbf{F}_{\text{mag}} \)" and the unit vector are combined to calculate the vector force "\( \mathbf{F}_{\text{net}} \)." (See figure 4.5 for an example of this procedure.) These steps could, of course, be combined into one line; the procedure is broken up to reduce the amount of cognitive load on the student at each step, but the drawback is the large number of steps that students must remember.

```plaintext
while t<28*24*60*60
    r = earth.pos - moon.pos
    rmag = sqrt(r.x**2 + r.y**2 + r.z**2)
    rhat = r/rmag
    Fmag = G*earth.m*moon.m/rmag**2
    Fnet = Fmag*rhat
```

*Figure 4.5.* Example of the procedure used to calculate the gravitational force vector.

Several students from both phases had difficulties with the vector nature of the force. A common approach was for students to create forces that were scalar expressions, seemingly without any consideration for the vector nature of the force, and then attempt to use this force to update the momentum. Kyle, for example, defined before the loop the
distance between the earth and the moon to be "moon.rmag=3.8e8", a scalar value.

Inside the loop, he defined his force: "Fnet = 6.7e-11 * (moon.m * earth.m)/moon.rmag**2", another scalar quantity, since the masses and the distance are all scalars. Kyle went on to write an expression for updating momentum and updating position. He had defined the initial momentum to be a scalar as well, so his momentum update line, "moon.p = Fnet*deltat + moon.p" did not cause an error when the program was run. But his position update line, "moon.pos = moon.v*deltat + moon.pos" did produce an error—his velocity "moon.v" was defined in terms of his scalar "moon.p", so it was also a scalar, but moon.pos was a vector. Later, Kyle corrected his initial momentum and velocity to make them vectors, but then his momentum update line caused the "adding a vector and scalar" error. Kyle then realized his force must be a vector, and redefined the magnitude of the force as "Fmag = 6.7e-11 * (moon.m * earth.m)/moon.rmag**2". But then he created a vector force that only points in one direction, never changing in the loop, by writing "Fnet = vector(0, -Fmag, 0)".

Kyle then struggled with the idea that a new direction for the force must be calculated each time through the loop. I did not intervene to correct this mistake right away. Since his program did run, but produced an incorrect path for the moon, I decided to see if he could correct it on his own. (His error was compounded by an incorrect initial momentum direction.) After a while, he became even more confused; he even tried using the momentum principle to calculate the force. At this point I said:

I: When we use that line [the momentum principle] we have to first have defined F net so that p will be updated. Do you remember how we defined F net, so that it's always pointing towards the earth?

S: You take the, you take the uh, final position minus the initial position.
I: Yeah. Yeah, that's gonna be involved.
S: And I need to define, or I can say earth dot pos, minus moon dot pos.

Kyle types: \( \mathbf{F_{net}} = \mathbf{earth.pos} - \mathbf{moon.pos} \)

So Kyle finally realized that the relative position vector is involved, but he defined the force to be equal to this vector. It took more explanation from me for him to see that this was not the force, but only a vector in the direction of the force. He then changed the force to be \( \mathbf{F_{net}} = (\mathbf{earth.pos} - \mathbf{moon.pos}) \cdot F_{mag} \), which is again incorrect—he did not realize that multiplying by the \( \mathbf{r} \) vector changes the magnitude of the force. After fruitless hints, I finally decided to show a written numerical example that explained why this formula was incorrect, and how instead one needed to use a unit vector called \( \mathbf{r_{hat}} \). Kyle immediately recognized \( \mathbf{r_{hat}} \) and remembered how to calculate it, but he never seemed to realize its necessity or purpose in the force calculation.

Other students had similar difficulties. Andrew also defined the force to be a scalar and ran into trouble with vector-scalar addition errors in the momentum principle. Upon realizing that he needed a vector expression, like Kyle he tried to multiply the magnitude of the force by \( \mathbf{r} \) instead of \( \mathbf{r_{hat}} \). I had to go through a similar example calculation as I did with Kyle to explain the role of \( \mathbf{r_{hat}} \) to Andrew.

Unlike Kyle, Paul remembered the need to define \( \mathbf{r} \) as \( \mathbf{r} = \mathbf{earth.pos} - \mathbf{moon.pos} \), and to calculate \( F_{mag} \), both in the loop before the force calculation. But then he wrote as his expression for the force \( \text{force} = G \cdot (\mathbf{moon.m} \cdot \mathbf{earth.m}) / \mathbf{r}^2 \). This is an invalid expression because it attempts to both divide by and square a vector quantity, which are undefined mathematical operations. He later corrected the force to be \( \text{force} = G \cdot (\mathbf{moon.m} \cdot \mathbf{earth.m}) / F_{mag}^2 \); which is valid, but is a scalar expression. Later, when he ran into vector-scalar addition errors in the momentum principle, he first
tried to solve this, like Kyle, by creating a force vector in a single direction: 
"force = (force, 0, 0)". He was, however, able to see this error without intervention from me.

Bobby, like Paul, first divided by and squared the relative position vector in his force expression. With minor prompting, he remembered he instead should divide by the magnitude of \( \mathbf{r} \). He remembered that an \"rhat\" appeared in the program, but did not know how to calculate it or how to use it without discussion.

Norman had difficulties with the force expression that seem to show a serious misunderstanding of either the physics concept of gravitational force, the mathematics of vectors, the nature of computer modeling, or all three. After starting a \"while\" loop, he decided to begin working on the force, and stated that it should be in the direction of the moon's position. He knew the force should be a vector, and he looked up the syntax for vectors in the VPython documentation. This help document displays the syntax for defining the numeric components of a vector by writing, for example 
\[ v = \text{vector}(x, y, z) \]. Perhaps this unduly influenced him or led him to forget about symbolic vector expressions, for he then said:

S: Ok, so I just have to put the uh, the three numbers in. Alright. Well for this um, I'm assuming that the position of the moon, is uh....is a good vector.

Next he typed "force = (3.8e8,0,0)\", which is the vector, written numerically, for the initial position of the moon. I began a discussion with him on whether the direction of the force changes, but this only led him to change the expression to "force = (moon.pos,0,0)\", a symbolic, yet invalid, version of his previous vector. Next I began to discuss with him symbolic vector expression and relative positions, which led him to write "force = vector(earth.pos-moon.pos)\". I finally told him that this was not the force, but "\( \mathbf{r} \)\", and he renamed it as such. This seemed to jog his memory, and he then
recalled the expression for \( \text{rmag} \) and \( \text{rhat} \). But he still seemed to be set on equating the force to a position vector, and did not realize the purpose of \( \text{rhat} \):

I: So you've got a vector that points in the correct direction, and has a magnitude of one. If I could multiply that by some other magnitude...

S: Like that magnitude [points to \( \text{rmag} \)]

I: Like that magnitude or in this case, what, what particular thing do we want to find?

S: The force.

I: The force, right. And it has to be in the same direction as....?

S: As, as the force.

I: Well the...

S: Or, or as, as the unit vector.

I: That's right. Force has to be in the same direction as this unit vector we just calculated, so to get a vector force that points in that direction, what should I do?

S: Well just the force equals the vector...force equals vector \( \text{rhat} \)? Will that work?

At this point, I explained that he needed an expression for the magnitude of the force, and he was able to correctly remember it. He still struggled with determining how to make a vector force that points in the correct direction. I had to show a simple example calculation of using a unit vector to make a new vector that points in the same direction but with a different magnitude from the original vector. This was able to make him recognize that he should multiply \( F_{\text{mag}} \) by \( \text{rhat} \).

As with the students' difficulties with time issues and initial velocity calculations, the fact that the problem involved a circular orbit may have caused or exacerbated students' difficulties with the force. The force magnitude is a constant in a circular orbit, so it may have led students to incorrect extrapolate that the direction is also constant. The fact that the magnitude of \( r \) does not change may have allowed students to believe there was no reason to calculate an \( r \) vector. It would be interesting to observe if similar difficulties result if students are given a problem with a generic initial velocity that states explicitly that the orbit is non-circular.
4.5.5 Momentum update issues

The momentum principle is the most fundamental physical principle emphasized by the Matter and Interactions course, and it plays a central role in the computer modeling of the motion of objects. Despite this, many students in both phases of the experiment had some difficulty with the momentum principle or momentum update statement in session 2, including some students who otherwise performed quite well on rest of the program. These difficulties were in addition to the vector issues mentioned in the previous sections. They included a lack of recall of the momentum principle, confusion with the momentum definition, and confusion with the position update statement.

David (from phase 2), for example, performed quite well in defining initial conditions, creating a loop, and calculating a gravitational force. But he began to have some trouble with the next step, updating the momentum. He said:

S: So now I need to update momentum of the moon, so moon dot p is going to be equal to the initial moon dot p plus, um...I wanna say it's gonna be F, but I know that's not correct.

He then wrote in the program "moon.p = moon.p + F", forgetting the factor of $\Delta t$ in the second term. Next he began to confuse updating momentum with updating position:

S: 'Cause I know somewhere we have to do momentum divided by mass to get velocity, and then, velocity times time, $\Delta t$.

This led him to alter the line to read "moon.p = moon.p + (moon.p/moon.m)*$\Delta t$"; the second term on the right hand side is the change in position. He quickly realized this mistake:

S: Oh hold on, this might be for...that'll give me the change in position.
He then rewrote the line to be the correct position update statement "moon.pos = moon.pos + (moon.p/moon.m)*deltat", but reinserted his incorrect momentum update line "moon.p = moon.p + F". When the program was run and gave anomalous behavior, he began to debug his force calculation. I then gave him a hint as to the location of the error:

I: You were confused about the line after it, right? The moon dot p equals moon dot p plus F.
S: Right. Oh it's F times delta t. Yeah because that's the change in, yeah change in momentum equals F net delta t.

This hint was enough to get him to focus on the momentum update line and see the missing deltat.

Charles also confused updating momentum and position. After writing a force law he next decided to "update the moon's position," and he typed "moon.pos = moon.pos + Fnet*deltat". Later, after debugging a variety of errors, mostly having to do with his force calculation, he tried to determine why the moon kept rapidly flying away from the earth. He kept focusing on the force calculations, which were correct by this point, so I hinted that something else may be a problem:

I: So let's say that, ok we know we're pretty confident the force cal-- the force calculation is correct. What else could be a problem here?
S: Updating the momentum?
I: Ok.

He then wrote a new line—the correct momentum update line, "moon.p = moon.p + Fnet*deltat", but his old, incorrect position update remained. After running the program again, and seeing that it still did not work, I pointed out that there may be a problem in the line following the momentum update. He then realized the mistake and corrected the position update statement.
Two students from phase 1, Paul and Bobby, made an error that is commonly seen in the Matter and Interactions course—ignoring or forgetting the "delta" symbol in the momentum principle. Many students often write that momentum is equal to force times time step, rather than change in momentum is equal to force times time step. Paul had a force calculation and next wanted to update the position, but struggled how to do it. First, he said "Force is momentum times time," then wrote "force = mv", but quickly erased this. After some thought he said:

S: So the resultant is the velocity times the time. That's how much, that's how much r changes each time. And the velocity is calculated from p equals m v, which is just p over m, and p is equal to F net times delta t, so that's F net times delta t over m times time.

But he wrote "moon.v=force*deltat". After some discussion, he changed it to "moon.p=force*deltat." Later in the session, after he corrected other problems, we came back to this issue. Only after a long discussion where I pointed out that "force*deltat" was the change in the momentum, and that he had to find a new momentum based on it, did Paul realize that he needed to update the momentum.

At the point where Bobby was going to add the momentum principle to his program, he had trouble recalling it:

S: And momentum is force times...times...force times distance?

This is actually the definition of work, a concept that had been just started in class at the time Bobby participated in session 2. After I pointed out that this was work, not the momentum principle, he realized it was force times deltats, but like Paul, he typed "moon.p = moon.F*deltat". I gave several hints about this mistake, but to no avail—I eventually just pointed out to him that it was the change in momentum, not momentum, that was force times change in time.
Nick had an interesting, yet invalid, approach that did not appear in any other student's protocol—he tried to combine updating momentum and position into one step. He first wrote "moon.pos = moon.pos + Fnet*deltat", but quickly realized that this was incorrect, saying "No, this is momentum." Next he worked on scratch paper, and starting with the momentum principle and the definition of momentum, he wrote:

\[ v_2 - v_1 = \frac{F_{net} \Delta t}{m} \]

Next, he tried to use the definition of velocity, \( \bar{v} = \Delta \vec{r} / \Delta t \), to write the velocities in terms of positions. But he wrote instead:

\[ \frac{r_2 - r_1}{\Delta t} = \frac{F_{net} \Delta t}{m} \]

Notice that either this means he believed \( v_1 \) was zero, or that the change in velocity is the change in position divided by time. After this, he solves for \( r_2 \):

\[ r_2 = r_1 + \frac{F_{net} \Delta t^2}{m} \]

He then translated this expression into his program, typing:

\[ \text{moon.pos}=\text{moon.pos}+\text{Fnet*deltat}**2/\text{moon.m} \]

He was, however, unsure of this approach, and even asked "Am I doing something wrong?" Later, after seeing an incorrect orbit upon running the program, he tried to debug issues such as the size of \( \text{deltat} \) and the details of the force calculation, but these were all correct. I decided to focus his attention on his incorrect position update statement:

I: You mentioned before you were having, you were suspicious of whether you were doing something wrong when you were trying to figure out that line [moon.pos update line].
S: Yeah.
I: What, what bothered you about it, I guess, what did you suspect might be wrong when you were doing it?
S: Mm, 'cause this doesn't seem familiar. [laughs]
I: Ok, so it doesn't look like you've done it before.
S: Yeah.
I: Can you think about...
S: Ok, yeah, I think I know what is wrong.

At this point, he is able to come up with correct, separate lines that update the momentum, and then the position.

4.6 Quantitative analysis of the dialogue

Since students from phase 2 of the experiment received a different instructional treatment, a quantitative analysis to compare the results of the two phases was performed. Because of the complicated nature of the dialogue between the students and the interviewer, a challenge arises in deciding how to quantitatively measure student difficulty. I decided to use my own verbal utterances for this purpose. My assumption is that the more often I had to intervene during a student's programming session, the more difficulty he was having with the program. Counting the number of interviewer interventions could serve as a rough measurement of student difficulty with a program and allow for comparisons between the two phases. Coding these utterances by the amount of help given and the topic being discussed could further indicate what parts of computer modeling are most difficult. Despite this unusual focus, the examination of the protocol data from sessions 1 and 2 of both phases of the experiment still followed Chi's method of verbal analysis, as described in section 2.3.4, and adhered to a slightly condensed form of the eight-step verbal analysis guidelines. (Verbal analysis was not used for the sorting task in session 3.) Furthermore, as discussed in section 2.3.3, the
verbalizations of tutors can be a useful source of data for analysis.

The first step in Chi's guidelines is reducing the protocols. For completeness, all data from all sessions were transcribed, but not all of the transcribed utterances were analyzed. Some students, due to time constraints, were not able to complete every part of problem statement. Therefore, in order to make fair comparisons between students, the students' verbal protocols were analyzed only up to the point where they all had completed a common amount of the program. For example, some students did not complete parts (d) and (e) in program 2 (moving rock) from session 1, where they were asked to make a trail to represent the rock's path and an arrow to represent its velocity. The verbal protocols were therefore coded and analyzed only up to the point where the students reached part (c); that is, the point where the program only displayed a sphere, representing the rock, moving at a constant velocity for 10 time steps. (I ignore the case of Richard, who did not have time to work on the moving rock problem at all.) Similarly, in the moon orbit program, the original problem statement asked students to make several alterations to the program once they had a functioning model of the orbit, such as adding a trail, adding arrows to represent force and momentum, and changing the initial velocity. Since many students did not reach these exercises, the protocols were analyzed only up to the point where students had written a program that only displayed a sphere for the moon revolving in circular orbit around a sphere for the earth. For more details on what parts were left out of the analysis, see section 7.3 in the Appendix.

As discussed in section 4.5, the data from session 2's moon orbit program is of particular interest for the purposes of comparing performance on a typical computer model of a physical system. As I will describe below, two coding schemes will be used
Step 2 of the guidelines is to segment the protocols into units of analysis. Verbal protocols can be analyzed at a variety of different grain sizes, from the paragraph level, to the sentence level, to the word level. When transcribing spoken language, there is ambiguity as to where exactly a sentence or paragraph begins and ends. To avoid this ambiguity, I decided to count each line of typewritten dialogue as one unit of analysis. Specifically, the dialogue was transcribed in 10 point font to standard 8.5 x 11 inch pages with one inch margins on all sides. Any line of dialogue that occupied one line in this format was counted as one unit of analysis. While this does make the counting and coding of dialogue more consistent, it does introduce the disadvantage of possibly "over-counting" the amount of interviewer interventions. For example, a statement consisting of just one or two words is counted as one line, just as a full page-width line would be. The number of these shorter lines can also be artificially inflated by incidents of "cross-talk" between the interviewer and the student; that is, if the student says something before the interviewer finishes a sentence, that sentence may be broken up into two segments in the transcript, with the student utterance between the two.

The next step is to develop a coding scheme. Two coding schemes were used. The first scheme was used to code all interviewer dialogue according to the extent to which the interviewer intervened in the student's work. Verbal protocols of both session 1 and 2 were analyzed with this scheme. I shall refer to this coding scheme as the "type of intervention." The four categories used in this scheme were the following:

1. Syntax. These comments refer to the specifics of VPython syntax, including

   • placement of parentheses, spaces, and colons;
• the names of VPython graphical objects (such as "sphere" or "arrow");
• any minor spelling mistakes.

Although students inevitably learn some VPython syntax, teaching minute syntax details was not a goal of the *Matter & Interactions* course, nor was examining student difficulty with such details a goal of the study. These interventions were judged to have minimal affect on students' reasoning with the more important physical and mathematical concepts in the program.

2. *Minor interventions.* These comments are minor acknowledgments or affirmations after a student takes a small step in a task or asks a question. They are mostly various ways of saying "OK, keep going." Examples of interviewer phrases that fall into this category include "Yes", "Mm-hm", "Uh-huh", and "That's right". Also, any line which repeats a student's statement as a way of acknowledging or affirming it is coded in this category.

3. *Major interventions.* These were comments that were judged to be most influential on students' reasoning during their programming. Remarks in this category could do any of the following:

• Point out to the student that an error or mistake exists in the program;
• Focus the student's attention on a particular problematic area, or on what step to take next;
• Offer hints or helpful information;
• Elaborate on or explain a student's work;
• Give a specific answer or solution to a step of the program, usually after a series of hints has failed.
4. **Other.** Comments that did not fall into any other categories were put into this category.

In order to be sure that I applied the coding scheme consistently to all of the protocol data, I measured inter-rater reliability on a subset of the data. One physics professor and one physics graduate student volunteered to each code approximately 2-hours worth of protocol data. The two raters each coded different data, so that the total amount of rated protocols was 4-hours out of a total of about 27 hours of data. The raters were given the above categories and coding criteria and coded the data without any other input from me. A ratio of the total number of lines where my coding agrees with the rater's to the total number of lines being examined yields an inter-rater reliability value of 0.90. Because of this high value, I am reasonably confident that I applied my coding scheme consistently to the rest of the data.

Because of the mentioned importance of session 2, a second additional coding scheme was used to analyze this session only, in addition to coding the interviewer dialogue in this session by the type of intervention. This second coding scheme was used to code by the topic of the intervention. This was done to find which specific parts of the program proved most difficult for students, and to search for any improvements on these parts from the first to the second semester. The protocol data analyzed using this scheme was a subset of the interviewer utterances that were coded in the first scheme. Only those utterances that were coded as major interventions were analyzed in this second scheme. This is because I assume that the major interventions were the utterances that were most indicative of conceptual or procedural difficulties in creating a computer model. Again, I
also assume that more interviewer utterances on a topic indicate more student difficulty
with the topic.

The second coding scheme divided the utterances into categories based on the
major steps needed to complete the computer model of the moon orbiting the earth. The
categories were:

1. *General probe or observation.* Any statement where the interviewer tries to
   probe what the student thinks is wrong without asking about a specific topic
   or part of the program. Also, any statement where the interviewer tells the
   student that the running program behaves incorrectly, but does not say why.

2. *Loops and program structure.* Statements that ask the student about what the
   program has to do in the next step, the need for a loop, the contents of the
   loop, or the quantities that change in the program.

3. *Time and time step.* Statements that pertain to:
   - confusion between t and delta t;
   - confusion between initial time and final time;
   - the need to update time (e.g. interviewer points out that the program
     continues without stopping); or
   - confusion between time and rate

4. *Force vector.* Comments that refer to any of the steps needed to calculate a
   gravitational force vector. This includes calculations for:
   - the relative position vector (r);
   - the magnitude of this vector (rmag);
   - the unit vector that gives the force direction (rhat);
• the magnitude of the gravitational force ($F_{mag}$); and

• the final force vector ($F_{net}$).

5. Initial velocity/momentum. Comments that refer to any of the steps needed to find the initial velocity or momentum, including:

• the need or purpose of initial conditions;

• calculation of the intial speed, $2\pi r / T$;

• finding the correct direction for the initial velocity (or momentum).

6. Momentum principle. Comments that specifically refer to the updating of the momentum by using the momentum principle in its discrete form: $\Delta \mathbf{p} = \mathbf{F} \Delta t$, where $\Delta t$ is a time interval over which the force can be approximated as constant.

7. Momentum definition. This refers to confusion with knowing or using definition of momentum, $\mathbf{p} = m \mathbf{v}$ (for low speeds). These statements usually occur when the student is trying to get an initial velocity from an initial momentum or vice versa, or trying to define calculate as $\mathbf{p} / m$ in the position update statement.

8. Position update. Comments that specifically refer to the updating of the position using the new velocity or momentum.

9. Other. Any statements which do not fit into the above categories.

Inter-rater reliability was again calculated for this second coding scheme. A physics graduate student volunteered to code 2-hours worth of protocol data according to the above guidelines. The rater coded only those lines that had been determined by me to be major interventions according to the first coding scheme. Because of the high inter-
rater reliability of the first coding scheme, I was reasonably confident that the data the
rater coded were indeed major interventions. The inter-rater reliability value for this
second scheme is 0.83. While smaller than that of the first coding scheme, it is still an
acceptable value. The lower value may be due to the larger number of categories.

The fourth step in Chi's guidelines is operationalizing evidence in the coded
protocols. Essentially, this means deciding which coded utterances constitute evidence of
the phenomenon in question. Since we are looking for evidence of improved student
performance in computer modeling, especially any improvement that might be
attributable to the instructional intervention, I will focus first on differences between the
two semesters in the number of major interventions given to students. Since these are the
comments that most affect student thinking, fewer of these given in a session might
indicate less student difficulty with the programming task. Operationalizing evidence also
typically involves creating a graphical concept map of the phenomenon, but considering
that it is the interviewer's utterances that are being examined, this did not seem
appropriate. Likewise, the optional stage of depicting the mapped formalism will be
skipped also.

The next two steps of Chi's guidelines, involving seeking and interpreting patterns
in the coded data, will be developed in the following sections. In section 4.7, I will
present the results of coding by type of intervention in session 1. In section 4.8, I will
present the results from both coding schemes for session 2. In reporting the results,
students in the study are identified by pseudonyms. In each case, analysis begins at the
point where the student formally begins working on the problem in question, and ends at
the point where the student has a functioning program for that problem. Briefings and debriefings by the interviewer are omitted from all analyses.

In section 4.9, I will report the quantitative results of the program line sorting task that students worked on in session 3 of the experiment. Refer to that section for details on the analysis.

An important question that arises when examining the quantitative verbal analysis results in the following sections is whether these results are generalizable to all students from the two M&I classes. This is a difficult question to answer due to the small sample sizes. For such an intensive study as this one, it is impossible to work with large numbers of participants. Few students volunteer, and even if more did, only a small number of students could be accepted for practical reasons—there is limited time to analyze the large amount of data each student generates. Statistical tests for differences lose validity with such small samples, so it is impossible to say any differences between phases is significant. Nevertheless, the quantitative results coupled with the detailed qualitative description of student details can give a description of the basic difficulties students face and the decisions they have to make in computer modeling tasks, and they can provide information for the future revision of instruction. Furthermore, in section 3.10 I will present another visualization of intervention coding data from session 2 that may shed light on students' choices of solution path while working on a computer model.

4.7 Results from verbal analysis of session 1

As stated in section 4.5, any comparisons between the two phases on the tasks in session 1 will be flawed, due to the differences between the phases in the programming
tasks that the students performed (a model of a table and stools vs. a static model of Jupiter and its moons). Nonetheless, the first programming task's focus on the use of vectors and the creation of arrows is related to instructional efforts on these matters in the developed instructional materials. Because of this, results are presented here for completeness.

4.7.1 Results from program 1, session 1: arrow objects and position vectors

Although the first program that students worked on differed between the two phases, they both explored similar concepts; namely, creating relative position vectors with VPython arrow objects. Despite the differences, I will make comparisons between the two tasks, considering their similar purposes.

Tables 4.2 and 4.3 show the results of coding by intervention type for the phase one and two of the experiment, respectively. With the exception of Richard, all students took about 20 to 30 minutes to complete their programs. In phase 1, each student received about 30 major interventions, with the exception of Richard, who received 127. In phase 2, each student received slightly less than 30 major interventions, with the exception of Nick, who received only eight.

Table 4.2. Results for phase 1 (first semester), session 1, table and stools program

<table>
<thead>
<tr>
<th></th>
<th>Bobby</th>
<th>Norman</th>
<th>Paul</th>
<th>Richard</th>
<th>mean</th>
<th>st.dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>7</td>
<td>0</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>3.2</td>
</tr>
<tr>
<td>Minor interventions</td>
<td>19</td>
<td>21</td>
<td>17</td>
<td>39</td>
<td>24</td>
<td>10.1</td>
</tr>
<tr>
<td>Major interventions</td>
<td>33</td>
<td>34</td>
<td>29</td>
<td>127</td>
<td>56</td>
<td>47.5</td>
</tr>
<tr>
<td>Other</td>
<td>10</td>
<td>9</td>
<td>12</td>
<td>35</td>
<td>17</td>
<td>12.4</td>
</tr>
<tr>
<td>Total interventions</td>
<td>69</td>
<td>64</td>
<td>61</td>
<td>207</td>
<td>100</td>
<td>71.2</td>
</tr>
<tr>
<td>Time to complete (min.)</td>
<td>29</td>
<td>29</td>
<td>19</td>
<td>54</td>
<td>33</td>
<td>15.0</td>
</tr>
</tbody>
</table>
Table 4.3. Results for phase 2 (second semester), session 2, Jovian moons program

<table>
<thead>
<tr>
<th></th>
<th>Andrew</th>
<th>Charles</th>
<th>David</th>
<th>Kyle</th>
<th>Nick</th>
<th>mean</th>
<th>st. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>17</td>
<td>8</td>
<td>25</td>
<td>9</td>
<td>8</td>
<td>13</td>
<td>7.5</td>
</tr>
<tr>
<td>Minor interventions</td>
<td>37</td>
<td>23</td>
<td>26</td>
<td>29</td>
<td>25</td>
<td>28</td>
<td>5.5</td>
</tr>
<tr>
<td>Major interventions</td>
<td>27</td>
<td>29</td>
<td>24</td>
<td>25</td>
<td>8</td>
<td>23</td>
<td>8.4</td>
</tr>
<tr>
<td>Other</td>
<td>14</td>
<td>17</td>
<td>3</td>
<td>11</td>
<td>14</td>
<td>12</td>
<td>5.4</td>
</tr>
<tr>
<td>Total interventions</td>
<td>95</td>
<td>77</td>
<td>78</td>
<td>74</td>
<td>55</td>
<td>76</td>
<td>14.2</td>
</tr>
<tr>
<td>Time to complete (min.)</td>
<td>28</td>
<td>23</td>
<td>21</td>
<td>24</td>
<td>21</td>
<td>24</td>
<td>3.0</td>
</tr>
</tbody>
</table>

Figure 4.6 displays the number of major interventions in program 1 for each student. Notice the similarity between three of the phase 1 students and four of the phase 2 students. Only Richard and Nick, especially Richard, stand out. Figure 4.7 shows that there is a sizeable drop in the number of major interventions from phase 1 to phase 2, but this difference is largely due to the presence of Richard's and Nick's results.

![Figure 4.6. Major interventions in the first program (arrows and position vectors) from session 1. Phase 1 results are in black, phase 2 results are gray.](image)
Was the number of interventions unfairly inflated or reduced for any student as a result of inconsistent questioning by me, the interviewer? It is difficult to determine this, but there are some indications that say no. The number of interventions given per minute for the students ranges from 2.2 for Norman, to 3.8 for Richard. The interventions per minute for Richard is the highest, but it does not appear to be very different from those of some good performing students, like Andrew (3.4), Charles (3.4), or David (3.7).
Consider figure 4.8, which plots the total number of interventions versus the time of completion for each student in both phases. The regression line has a slope of about 4.1 interventions per minute. The regression line has a high R-squared value of 0.88, but unfortunately, this is largely due to the dependence on the extreme data point of Richard to determine the line. So while it is plausible that the questioning was consistently applied, it is not conclusive. In later programs, we will see that the consistency of the interviewer seems more plausible, because the number of interventions per minute among the students is more evenly spread.

4.7.2 Results from program 2, session 1: moving rock

Program 2 from session 1, the moving rock program, is the first program that was identical for both phases. Unfortunately, direct comparison of the two phases is again difficult for two reasons. First, Richard did not attempt program 2, due to lack of time in
the session. (His work on other programs suggests that he might have struggled with this program as well.) Second, Norman inadvertently used on an online worked-out example of how to update position in VPython. This was a resource provided to students in the course that should have been excluded from the interview sessions; I was not aware of its inclusion on the course web page until it was too late. It effectively showed him the correct procedure to use, and therefore likely reduced the amount of major interventions I would have made otherwise. Norman was the only student to use this resource, and it only occurred in this program.

Tables 4.4 and 4.5 show the results from phases 1 and 2 respectively. The number of major interventions varies greatly among the students, especially in phase 1, where it ranges from 1 to 57; in phase 2, it ranges from 17 to 66.

**Table 4.4.** Results for phase 1, session 1, moving rock program

<table>
<thead>
<tr>
<th></th>
<th>Bobby</th>
<th>Norman</th>
<th>Paul</th>
<th>Mean</th>
<th>St. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>2</td>
<td>7</td>
<td>0</td>
<td>3</td>
<td>3.6</td>
</tr>
<tr>
<td>Minor interventions</td>
<td>9</td>
<td>8</td>
<td>3</td>
<td>7</td>
<td>3.2</td>
</tr>
<tr>
<td>Major interventions</td>
<td>57</td>
<td>6</td>
<td>1</td>
<td>21</td>
<td>31.0</td>
</tr>
<tr>
<td>Other</td>
<td>10</td>
<td>15</td>
<td>1</td>
<td>9</td>
<td>7.1</td>
</tr>
<tr>
<td>Total interventions</td>
<td>78</td>
<td>36</td>
<td>5</td>
<td>40</td>
<td>36.6</td>
</tr>
<tr>
<td>Time to complete (min.)</td>
<td>20</td>
<td>10</td>
<td>3</td>
<td>11</td>
<td>8.3</td>
</tr>
</tbody>
</table>

**Table 4.5.** Results for phase 2, session 1, moving rock program

<table>
<thead>
<tr>
<th></th>
<th>Andrew</th>
<th>Charles</th>
<th>David</th>
<th>Kyle</th>
<th>Nick</th>
<th>Mean</th>
<th>St. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>9</td>
<td>18</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3.1</td>
</tr>
<tr>
<td>Minor interventions</td>
<td>14</td>
<td>7</td>
<td>19</td>
<td>27</td>
<td>10</td>
<td>19</td>
<td>8.5</td>
</tr>
<tr>
<td>Major interventions</td>
<td>31</td>
<td>29</td>
<td>17</td>
<td>66</td>
<td>18</td>
<td>34</td>
<td>28.0</td>
</tr>
<tr>
<td>Other</td>
<td>13</td>
<td>15</td>
<td>15</td>
<td>36</td>
<td>20</td>
<td>24</td>
<td>11.0</td>
</tr>
<tr>
<td>Total interventions</td>
<td>67</td>
<td>69</td>
<td>59</td>
<td>131</td>
<td>52</td>
<td>81</td>
<td>43.7</td>
</tr>
<tr>
<td>Time to complete (min.)</td>
<td>18</td>
<td>14</td>
<td>11</td>
<td>26</td>
<td>12</td>
<td>16</td>
<td>8.2</td>
</tr>
</tbody>
</table>
Figure 4.9 graphically compares the number of major interventions from all students, while figure 4.10 compares the average number of interventions for the two phases. Phase 2 had a larger average number of interventions in all categories. It might appear that the second phase had a greater difficulty with the moving rock program, but again, this is with no data from one phase 1 student and anomalous data from another.

Figure 4.9. Major interventions in the moving rock program from session 1. Phase 1 results are in black, phase 2 results are gray.
Figure 4.11 plots number of lines of interviewer interventions versus the amount of time to complete program 2. It shows that I appear to intervene consistently across all the students at about 5 intervention lines per minute for this program. The R-squared value for the regression line is 0.91.
4.8 Results from verbal analysis of session 2

Next I will present the results of the verbal analysis for session 2, the moon orbit program. Some students had enough time to go on to secondary exercises related to this problem (see parts b through h of the instructions for the orbit program in the Appendix); these were omitted from the analysis.

4.8.1 Results from the first coding scheme (type of intervention)

Tables 4.6 and 4.7 show the results of the first coding scheme for the first and second semester, respectively. The spread of scores for both phases is quite wide. Major interventions for phase 1 ranges from 72 to 292, from 18 to 165 for phase 2. Total interventions ranges from 160 to 258 for phase 1, and from 43 to 255 for phase 2. Again
Richard stands out in the time he took to complete the program and in his large number of major and total interventions.

Table 4.6. Results of coding by intervention type, phase 1 (first semester), session 2 (orbit program).

<table>
<thead>
<tr>
<th></th>
<th>Bobby</th>
<th>Norman</th>
<th>Paul</th>
<th>Richard</th>
<th>mean</th>
<th>st. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>10</td>
<td>15</td>
<td>15</td>
<td>20</td>
<td>15</td>
<td>4.1</td>
</tr>
<tr>
<td>Minor interventions</td>
<td>41</td>
<td>63</td>
<td>43</td>
<td>90</td>
<td>59</td>
<td>22.8</td>
</tr>
<tr>
<td>Major interventions</td>
<td>114</td>
<td>158</td>
<td>72</td>
<td>292</td>
<td>159</td>
<td>95.1</td>
</tr>
<tr>
<td>Other</td>
<td>23</td>
<td>22</td>
<td>30</td>
<td>59</td>
<td>34</td>
<td>17.4</td>
</tr>
<tr>
<td>Total interventions</td>
<td>188</td>
<td>258</td>
<td>160</td>
<td>461</td>
<td>267</td>
<td>135.9</td>
</tr>
<tr>
<td>Time to complete (min.)</td>
<td>50</td>
<td>59</td>
<td>32</td>
<td>76</td>
<td>55</td>
<td>18.5</td>
</tr>
</tbody>
</table>

Table 4.7. Results of coding by intervention type, phase 2 (second semester), session 2 (orbit program).

<table>
<thead>
<tr>
<th></th>
<th>Andrew</th>
<th>Charles</th>
<th>David</th>
<th>Kyle</th>
<th>Nick</th>
<th>mean</th>
<th>st. dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>12</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>3.9</td>
</tr>
<tr>
<td>Minor interventions</td>
<td>54</td>
<td>40</td>
<td>11</td>
<td>49</td>
<td>13</td>
<td>33</td>
<td>20.2</td>
</tr>
<tr>
<td>Major interventions</td>
<td>165</td>
<td>51</td>
<td>18</td>
<td>136</td>
<td>21</td>
<td>78</td>
<td>68.0</td>
</tr>
<tr>
<td>Other</td>
<td>24</td>
<td>20</td>
<td>11</td>
<td>37</td>
<td>7</td>
<td>20</td>
<td>11.8</td>
</tr>
<tr>
<td>Total interventions</td>
<td>255</td>
<td>116</td>
<td>48</td>
<td>226</td>
<td>43</td>
<td>138</td>
<td>98.8</td>
</tr>
<tr>
<td>Time to complete (min.)</td>
<td>58</td>
<td>31</td>
<td>21</td>
<td>52</td>
<td>29</td>
<td>38</td>
<td>16.0</td>
</tr>
</tbody>
</table>

Figure 4.12 is a chart of all the major interventions for all the students. One striking feature of the data as seen in this chart is that three of the five students from phase 2 had 51 major interventions or fewer, while the lowest number of major interventions for students from phase 1 was 73. Also, the largest number of major interventions from phase 2 (165 for Andrew) is much lower than the largest number from phase 1 (292 for Richard).
Figure 4.12. Major interventions in the orbit program. Phase 1 student data are in black, phase 2 results are in gray.

As shown in Figure 4.13, on average there were fewer interventions in all categories in the second semester than there were in the first. In phase 2, both the mean number of major interventions and total interventions dropped to about half of their values from phase 1.

Figure 4.14 plots the total number of interventions against the time to complete the moon orbit program for each student. Again it suggests that interventions were applied consistently across students, at roughly a rate of 7 interventions per minute. The R-squared value for the regression line is 0.91.
Figure 4.13. Comparison between the two phases of the mean number of types of interventions, session 2.

Figure 4.14. Total interventions versus completion time for session 2 (moon orbit program).
4.8.2 Results from the second coding scheme (topic of intervention)

Tables 4.8 and 4.9 show the results of the second coding scheme from the first and second semesters, respectively. Figure 4.15 compares the mean number of interventions by topic for both phases of the study. For each topic, the mean number of interventions dropped in phase 2. Both this category and "Loops and program structure" stand out in their marked reduction in semester 2 as compared to the others. While the mean number of all major interventions went down for semester 2 by 51%, the mean number of interventions in these two categories each reduced by more than 90%. All other categories (except "Other") saw reductions that ranged from 28 to 61% (see table 3.5).

Table 4.8. Results of coding by topic of intervention, phase 1, session 2.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Bobby</th>
<th>Norman</th>
<th>Paul</th>
<th>Richard</th>
<th>mean</th>
<th>st.dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>General probe or obs.</td>
<td>12</td>
<td>5</td>
<td>17</td>
<td>8</td>
<td>11</td>
<td>5.2</td>
</tr>
<tr>
<td>Loops, program structure</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>24</td>
<td>9</td>
<td>11.6</td>
</tr>
<tr>
<td>Time and time step</td>
<td>3</td>
<td>9</td>
<td>2</td>
<td>46</td>
<td>15</td>
<td>20.9</td>
</tr>
<tr>
<td>Force vector</td>
<td>33</td>
<td>96</td>
<td>11</td>
<td>74</td>
<td>54</td>
<td>38.5</td>
</tr>
<tr>
<td>Initial velocity/momentum</td>
<td>28</td>
<td>30</td>
<td>25</td>
<td>35</td>
<td>30</td>
<td>4.2</td>
</tr>
<tr>
<td>Momentum principle</td>
<td>15</td>
<td>0</td>
<td>14</td>
<td>51</td>
<td>20</td>
<td>21.8</td>
</tr>
<tr>
<td>Momentum definition</td>
<td>8</td>
<td>6</td>
<td>0</td>
<td>13</td>
<td>7</td>
<td>5.4</td>
</tr>
<tr>
<td>Position update</td>
<td>2</td>
<td>12</td>
<td>3</td>
<td>39</td>
<td>14</td>
<td>17.3</td>
</tr>
<tr>
<td>Other</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Table 4.9. Results of coding by topic of intervention, phase 2, session 2.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Andrew</th>
<th>Charles</th>
<th>David</th>
<th>Kyle</th>
<th>Nick</th>
<th>mean</th>
<th>st.dev.</th>
<th>% red.</th>
</tr>
</thead>
<tbody>
<tr>
<td>General probe or obs.</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>7</td>
<td>8</td>
<td>2.6</td>
<td>27.6</td>
</tr>
<tr>
<td>Loops, program structure</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.8</td>
<td>91.4</td>
</tr>
<tr>
<td>Time and time step</td>
<td>16</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>6</td>
<td>5.9</td>
<td>57.3</td>
</tr>
<tr>
<td>Force vector</td>
<td>71</td>
<td>27</td>
<td>4</td>
<td>56</td>
<td>0</td>
<td>32</td>
<td>31.3</td>
<td>40.9</td>
</tr>
<tr>
<td>Initial velocity/momentum</td>
<td>44</td>
<td>3</td>
<td>1</td>
<td>32</td>
<td>0</td>
<td>16</td>
<td>20.6</td>
<td>45.8</td>
</tr>
<tr>
<td>Momentum principle</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td>14</td>
<td>6</td>
<td>8</td>
<td>4.9</td>
<td>61.0</td>
</tr>
<tr>
<td>Momentum definition</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0.5</td>
<td>94.1</td>
</tr>
<tr>
<td>Position update</td>
<td>7</td>
<td>4</td>
<td>0</td>
<td>18</td>
<td>2</td>
<td>6</td>
<td>7.1</td>
<td>55.7</td>
</tr>
<tr>
<td>Other</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1.9</td>
<td>-180.0</td>
</tr>
</tbody>
</table>
4.9 Results of session 3 line-sorting task

In the first two sessions of the study, students had a great deal of difficulty with the details of constructing a computer model. The possibility still existed that students may understand the basic structure and organization of a computer model (i.e. initial conditions and constants followed by a dynamics loop), even if they could not construct all the details without help. The task in session 3 was designed to examine this possibility. Session 3 came late in each semester, after the students had been exposed to all of the computer modeling activities in the course.

Figure 4.16 shows the lines of a VPython program arranged in a random order. These lines were broken into nine distinct "chunks" and displayed in a VPython editor window. The chunks were chosen by the conceptual step that they referred to in the modeling procedure; for example, using the gravitational force law, updating the
momentum using the momentum principle, declaring an initial momentum, creating objects at initial positions, or calculating a unit vector to use for the force direction. A second editor window contained only a skeleton of a program that consisted of the initial line "from visual import *" followed by white space and the "while" statement (see figure 4.17). Students were only told that if the chunks of lines were arranged in the correct order within the second editor window, they would make a program which models a planet orbiting a star. They were then asked to correctly arrange the chunks by copying and pasting them into second editor window, and to run the program whenever they wished to test their work.
Figure 4.16. Randomly arranged lines of a program, used in the sorting task in session 3. Students copied and pasted the lines into the "skeleton" of a program, shown in figure 4.17.
Figure 4.17. The outline of VPython program from session 3. Students copied and pasted randomly arranged chunks of line, seen in figure 4.16, into this "skeleton" to make a working computer model.

As measure of performance, I counted the number of steps students took to complete this task. A step could either be a "move" or a "run". A move was counted whenever a student copied one of the chunks of lines and pasted it into the program file, or whenever a chunk was moved to a different point within a program. A run was counted whenever the student ran the program. The minimum number of steps a student could take to successfully complete the task is ten; nine moves to copy and paste each chunk in the correct order, and one run to verify the working program. (Indenting a line was not considered a move. Any student who placed a line underneath the "while" statement and did not indent it was reminded to do so.)

```python
from visual import *

while t<1.8e9:
```

172
Table 4.10. Results of session 3 sorting task, phase 1

<table>
<thead>
<tr>
<th></th>
<th>Moves</th>
<th>Runs</th>
<th>Total steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bobby</td>
<td>9</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Norman</td>
<td>11</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>Paul</td>
<td>12</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>Richard</td>
<td>18</td>
<td>8</td>
<td>26</td>
</tr>
<tr>
<td>mean</td>
<td>13</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>st.dev.</td>
<td>3.9</td>
<td>3.0</td>
<td>6.8</td>
</tr>
</tbody>
</table>

Table 4.11. Results of session 3 sorting task, phase 2

<table>
<thead>
<tr>
<th></th>
<th>Moves</th>
<th>Runs</th>
<th>Total steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrew</td>
<td>13</td>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>Charles</td>
<td>11</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>David</td>
<td>9</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Kyle</td>
<td>9</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Nick</td>
<td>9</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>mean</td>
<td>10</td>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>st.dev.</td>
<td>1.8</td>
<td>1.1</td>
<td>2.8</td>
</tr>
</tbody>
</table>

Tables 4.10 and 4.11 show the results of the sorting task. Nearly all the students could construct a working program in relatively few steps. Only Richard stands out once again as having much more difficulty with the task than others, otherwise the performance of both semesters is strikingly similar. One student in the first semester and three in the second completed the task in the fewest steps possible.

4.10 Transitions between program steps in Session 2

Since the interventions in session 2 were categorized by topic, based on the major steps of the program, this coded data can be used to visualize another aspect of students' progress during the session. Recall that there were nine major intervention categories
used in second coding scheme for session 2. Two of these categories ("general probe" and "other") referred to utterances that were not related to a specific step in the program. The remaining seven were:

1. Initial velocity and momentum
2. Loops and program structure
3. Time and time step
4. Force vector
5. Momentum principle
6. Momentum definition
7. Position update

The above list of categories reflects one possible order in which someone who is experienced in writing computer models in VPython might tackle the various stages of the program. Of course this order might vary; for example, one might decide to create the while loop first, then go back and declare the initial conditions. But in any case, someone who has had much practice in writing VPython computer models would likely deal with and complete each of these major steps one at a time. For example, the major subgoal of writing all the steps of the force calculation would be dealt with at one time before moving on to another major part of the program. Aside from possibly correcting syntax errors, we might expect little "cycling back" to previously completed steps.

This then leads to the question of students' solution paths in session 2: did they deal with the steps of the program in a straightforward way, one at a time? Did they "cycle back" to previous steps? Were there differences between students, and between the two phases of the experiment? And how can we measure this?
We can reasonably assume that the transcript lines coded by topic of intervention indicate what part of the program the student was working on at the time. Knowing this, we could plot the interventions by their category along a sort of "time" axis represented by the line number of the transcript. This can give us a visual guide to students' solution paths as they worked on the moon orbit program—it serves a sort of plot of the steps they worked on as a function of time. Because of the use of interviewer intervention data, these plots will not give a complete record of a student's solution path; students could have worked on different steps of the program where the interviewer did not intervene, for example. Despite this limitation, these plots may provide a qualitative, visual description of student progress.

Figures 4.18 through 4.21 show these plots for the students from phase 1; figures 4.22 through 4.26 show them for students from phase 2. The numbers on the y-axis correspond to the numbered topic categories listed above (e.g., 1 stands for initial velocity and momentum, 2 for loops and structure, etc.). The line segments between the data points are only a visual guide to transitions between steps; their length and slope do not have any meaning.
Figure 4.18. Bobby (phase 1): session 2 intervention category plot.

Figure 4.19. Norman (phase 1): session 2 intervention category plot.
Figure 4.20. Paul (phase 1): session 2 intervention category plot.

Figure 4.21. Richard (phase 1): session 2 intervention category plot.
Figure 4.22. Andrew (phase 2): session 2 intervention category plot.

Figure 4.23. Charles (phase 2): session 2 intervention category plot.
Figure 4.24. David (phase 2): session 2 intervention category plot.

Figure 4.25. Kyle (phase 2): session 2 intervention category plot.
Since the number of transitions between stages, and not the order of the stages themselves, is most important, we could also represent this data by simply plotting the number of transitions in the above graphs on a bar chart. Figure 4.27 shows the total number of transitions between intervention categories for each student. Essentially, this is the number of nonzero sloped line segments in the above plots. Figure 4.28 shows only the number of transitions back to steps of the program that were previously worked on.

**Figure 4.26.** Nick (phase 2): session 2 intervention category plot.
Figure 4.27. Number of transitions between intervention categories for each student. Phase 1 student data are in black, phase 2 results are in gray.

Figure 4.28. Number of transitions back to previous intervention categories for each student. Phase 1 student data are in black, phase 2 results are in gray.
Perhaps the most striking feature to emerge from these representations is that, roughly speaking, students who had more difficulty with the programming task, as measured by the number of major interventions and by qualitative analysis of their protocols, also seem to have the least straightforward solution paths, as measured by the "jumpiness" of these plots, or the number of transitions between steps of the program. Consider Richard as an example. He had the largest number of major interventions of any student, he had severe misunderstandings of the structure and purpose of the computer model, and he could remember very few details of the calculations without help. He also appears to have the greatest number of transitions between steps of the program. Contrast his plot with David's and Nick's from phase 2. They not only did well as measured by the small number of major interventions, but they also took very straightforward paths, without much cycling back to previous steps, to a complete program. Some students, like Norman from phase 1 and Charles from phase 2, did not seem to cycle between steps much, but did have a great deal of difficulty on one step, namely, the force vector calculation.

It is possible that the nature of the students' misunderstandings could contribute to their cycling between parts of the program. Kyle and Andrew, as discussed in section 3.9, often did not realize that quantities such as force and initial velocity needed to be vectors. Once error messages informed them of scalar-vector addition problems, it caused them to cycle back and fix problems with sections of the program they had already worked on. This may also be partly a result of the fact that I did not intervene until a student asked for help on a part of the program. This means that a student could make a logical or physical error in a program in a program without realizing it (for
example, declaring a syntactically correct but physically incorrect initial momentum), and then move on to the next step. Later, once it was clear that the model was behaving incorrectly, the student would have to go back and fix the earlier mistakes.

4.11 Discussion

While there are encouraging results from this study, we must be extremely cautious in claiming improvements in phase 2, not to mention the cause for those improvements. Results from session 1 notwithstanding, overall student performance as measured by interviewer interventions was better for the students in phase 2, but because of the small sample sizes, it is difficult to generalize this result. Furthermore, the improvement may be simply due to having stronger students volunteer in phase 2 than in phase 1.

In any case, both phases of the study have provided extremely detailed data on the nature of student difficulties with computer modeling. The results have also highlighted the fact that writing a computer model is a complex problem-solving task. To successfully compose a computer model of a physical system, a student must successfully complete a large number of tasks and subgoals, such as defining initial conditions, creating a loop, and calculating a force. One source of trouble may be the high cognitive load entailed by any complex problem-solving procedure. An additional source of cognitive load is the fact that often the students are called to generate certain inputs to the program, such as the time step or the initial momentum, that might be given to them if the problem were in a pencil-and-paper context.
The question arises as to whether the difficulties students experienced were with programming or with physics. It is difficult to separate the two, since the physics concepts that students need to understand are expressed through the formal language of programming. However, it appears that not all of the problems that students had in this study were specific to programming. Consider that the number of major interventions counted in session 2 on average was far greater than the number of interventions specific to syntax. These major interventions, as interpreted through the second coding scheme, largely had to do with physics concepts, such as force, the momentum principle, and initial conditions. These conceptual and mathematical misunderstandings were also seen in their written algebraic work on paper. Instructors in the Matter and Interactions course have often reported students having trouble with distinguishing vectors from scalars, ignoring the "delta" symbol, calculating relative position vectors and unit vectors, and confusing the momentum principle with the definition of momentum. It is only natural that these same misunderstandings would appear in students' work while programming in VPython.

What distinguishes programming from written algebraic work is that it forces students to think about and deal with physics difficulties that might be ignored or glossed over in written work. For example, incorrect initial conditions will cause a program to display a visibly incorrect orbit; a similar mistake on a pencil-and-paper problem does not give the same visual feedback. Similarly, students in their written work may make errors with forces similar to the ones seen in this study, such as writing force as a scalar quantity, treating it as constant in orbital motion, or dividing by a vector position in the gravitational force law. However, in written work, there is no feedback associated with
these mistakes, except perhaps corrections from a grader which a student sees much later. The same mistakes made in a program will produce feedback instantly visible to the student. For example, if force is defined to be a scalar, an error will occur when the program sees a vector momentum added to a scalar impulse, and the program will not run. If force is a constant vector in the moon orbit program, the moon will clearly not orbit the earth, giving instant visual feedback.

Furthermore, the programming environment provides a means to emphasize non-constant forces, which are central to the *M&I* course. Constant forces are often over-emphasized in introductory mechanics because non-constant forces can be too mathematically complicated to deal with. One goal of the *M&I* course is to help students see that most of the important forces in nature are not constant, and computer modeling provides a venue for introductory students to see this.

Other difficulties may at first glance have only to do with programming, such as difficulties with program structure, including aspects such as line order, the need for a loop, and the use of general principles instead of specific kinematics equations. But even these problems go beyond being specific to programming. They are fundamental to understanding the central physical process that at the heart of the *Matter and Interactions* curriculum; namely, that by knowing the force law and repeatedly applying the momentum principle over short intervals, we can predict the motion of any particle. The syntax of a program is specific to the particular programming language, and syntax errors were not my main concern; but the structure of the program serves as an analog for what really happens in nature, so it is extremely important for students to understand it. Recall
Sherin's main idea: that moving the study of physics to the language of programming emphasizes new concepts that cannot be explored through algebraic representations.

It is encouraging that there were fewer instances of problems with program purpose and structure in phase 2. We see that two students in phase 1, Bobby and Richard, had such difficulties. They were confused as to what types of statements need to be placed in the program, especially in the "while" loop. These confusions nearly disappear in phase 2; students know what statements need to go in the program and in the loop, even if they are still confused about the specific details of those statements.

Furthermore, in both phases, it seems that students over the course of the semester did recognize the structure and order of a program as measured in the line sorting task of session 3. Program structure was a topic that was specifically emphasized in the new instructional materials used in the second semester. No claim can be made that this alone made the difference, but it seems plausible that emphasizing the overall structure at an early stage could give students a better guide to program organization and help reduce one source of difficulty.

With regard to vectors and scalars, there is one small advantage that algebraic notation has over VPython syntax. In algebra, a vector quantity can be distinguished as a vector by the symbolic notation. Often a letter that represents a vector is written with a small arrow above it, and unit vectors are written with carets, or "hat" symbols. This type of notation is impossible with plain text, such as that which is used in VPython scripts. One way around this is to instruct students to use a naming convention for variables in VPython that indicates whether a quantity is a vector, such as typing an extra symbol, like an underscore, after a variable name. For example, a force vector might be called
"Fnet_", and a momentum might be "moon.p_", while a scalar like mass would be written without the underscore, such as "moon.m". (Naming conventions that indicate a variable's type are sometimes used by professional programmers; one such convention is called "Hungarian notation," created by Charles Simonyi of Microsoft Corporation.) One disadvantage is that vector attributes that are native to VPython objects, such as "pos" and "axis", do not use this convention, and therefore might cause confusion.

Another method that could help clear up not only confusions with vectors, but perhaps other difficulties, is to encourage students to write more comments in their programs. Comments are text statements within the code that offer descriptions and explanations of the variables and calculations used within the code. Comments are marked with a special symbol (in VPython, it's the pound sign, "#") that tell the computer to ignore the line; it's just text, not instruction. Students in Matter and Interactions rarely use comments. Requiring students to write comments that describe the nature and purpose of the variables and procedures in the program (e.g. "# moon.p is the initial momentum" or "# The next line updates the position") could be an effective learning tool. It might force student to think carefully about each part of the program, and discourage simply writing lines of code by rote. It could also serve as a valuable diagnostic tool; instructors could detect if students are having conceptual difficulties if they write incomplete or incorrect comments in their code. The drawback is that, like any skill, writing comments requires instruction, examples and modeling by instructors, and continual practice by students. This might take away time and resources from other important aspects of the course.
One of the main goals of this research is to uncover the problematic areas for Matter & Interactions students so that instruction for this expanding curriculum can be effectively revised. Already, the results of this research are being used for this purpose. For example, one important consequence of this study was that it showed just how complex a task it is for students to create a computer model of a physical system from scratch, even a relatively simple two body system. Constructing the force vector in particular is a complex, multistep procedure that involves multiple vector quantities, and it caused students great difficulty. Causing further problems for students is how to calculate and initial velocity that leads to circular motion. Both of these concepts were introduced to students at the same time, in the same programming assignment. This may have caused a high cognitive load; so many new ideas are introduced to students at one time that it becomes overwhelming. A better approach is to introduce one new idea at a time and allow students sufficient opportunity to practice it, so that they can assimilate it and then later integrate more easily with other concepts.

These lessons learned from this study have led Matter and Interactions instructors in Spring 2005 to revise the early laboratory sequence that introduces programming. The sequence now proceeds as follows:

- Lab 1: Students use the first part of the instructional materials from this study, which introduce them to VPython, arrow and sphere creation, and loops. In addition, students are introduced to updating position in this lab, as opposed to the second lab in the original instruction.
- Lab 2: Similar to lab 2 in the instruction created for this study. Students explore constant velocity and constant force motion with a fan cart. They then
model this motion with VPython, including a case where the initial velocity has a y-component. The momentum principle is the new concept here.

- Lab 3: In this new lab, students are introduced to the steps needed to calculate a gravitational force vector. Students have to write a program that calculates a force vector at multiple static locations near an asteroid, and display it with an arrow object, appropriately scaled. No motion is modeled, and no loops are used.

- Lab 4: The concepts introduced in the previous three labs are now brought together. Students write a program that models the motion of a planet around a star. The initial velocity is given, and it produces an elliptical orbit. After getting the program to run successfully, students try different initial velocities to see different trajectories. They are then asked to calculate an initial velocity that produces a circular orbit.

In this way, each major step of the program is introduced individually and practiced before they are integrated in the orbital motion model in lab 4. As of this writing, this instructional sequence has just been used, so there is no direct evidence of its success. However, instructors have reported that the sequence has gone smoothly; overall, students have performed each task successfully in the labs, with very little trouble.
5 Summary and conclusion

This dissertation has discussed student difficulties with computer modeling in the Matter & Interactions curriculum. Students have trouble both in deciding to use a computer model to solve a problem, as seen in experiment 1, and in the concepts and components of creating a computer model, as seen in experiment 2.

Matter & Interactions students, when approaching difficult problems that involve predicting future motion, either have difficulty realizing that such problems can be attacked with a computer model, or are uncomfortable with using a computer based approach. Those who did not write a computer model often still followed the dynamical algorithm of updating momentum, using the momentum principle in discrete form, followed by updating position. So overall, the M&I students made use of the key conceptual tool that they were taught in approaching these difficult problems. Traditional course students likewise made use of the tools and principles that were emphasized in their course, such as Newton's second law ($F=ma$), and certain special case formulas, such as those for circular motion or constant acceleration. The tools and approaches that the M&I students' used could be more readily adapted to computer modeling than those of the traditional students, but more work still needs to be done to convince M&I students that computer modeling is a useful and legitimate tool for attacking different problem situations.

When working on computer models, M&I students had a variety of difficulties with both the structure of the model and the physical principles that are used to construct the model. The complex procedure for calculating non-constant gravitational forces between bodies proved especially challenging for students. Students were able to
recognize the structural elements of a computer model later in the course. Students who had less difficulty with constructing a computer model also attacked the pieces of the model in a less circuitous, more straightforward way. Instruction early in the course was altered to help alleviate some problems with computer modeling issues. Students in the second phase, who had used the new instructional materials, did show improvement, especially in their understanding of the structure of the computer model and the purpose of loops, which the materials had emphasized. However, improvement cannot be attributed solely to this revised instruction. The revised instruction from this study has proved useful at NC State in quickly and effectively getting students acclimated to computer modeling in VPython. It has served an important role as a basis for instructional development, which is especially critical considering that M&I will be taught to all sections of introductory physics for engineering majors at NC State beginning in Fall 2005, and that other major universities are planning to adopt the curriculum.

Improvements could be made in instruction to help alleviate difficulties. Instructors may need to place more emphasis on the purpose of computer modeling at more frequent intervals during the semester. Homework problems could be given that encourage students to choose any means, including computer modeling, as a solution. Weekly homework assignments might also contain short problems or questions that test basic computer modeling skills. They might, for example, ask students how to write in VPython a particular vector position or given force law, or how to calculate an initial speed given certain conditions. Continual practice with these basic steps might reduce difficulty for students when the time comes to construct a full model of a new system.
Another possibility is to develop assessment tools such as multiple choice tests or quizzes that consist of such short questions. These can help instructors quickly gauge student difficulties on a class-wide basis relatively quickly, and they would give a better picture of how widespread the difficulties are than interviews with a small number of students. They may even give a more accurate picture of the students' current understanding of computer modeling than looking at the completed models themselves. The reason is that students' completed computer models are turned in after the students have had opportunity to consult instructors, teaching assistants, or other students for help on the assignments.

A wide variety of unanswered questions remain that could be pursued in further research. Although the question was examined to some extent in Experiment 1, we do not have a clear picture of how the Matter & Interactions curriculum affects students' problem solving skills and processes. A more extensive protocol analysis could be done on the data from Experiment 1, or a new protocol study could be performed that looked at other difficult problems that did not necessarily involve the prediction of motion or iterative methods. Areas to examine include overall problem solving strategy (e.g. means-ends, forward chaining), initial planning, and final evaluation or checking of a solution. Another question to be examined is whether students make explicit assumptions or approximations in complex problems. Making assumptions is a piece of the physical modeling process that is explicitly emphasized by the M&I curriculum.

With regard to computer modeling, one large unexamined area is the affect that 3-D visualization has on students' understanding of physical phenomenon. Experiment 2 concentrated primarily on student understanding of the algorithmic code of the computer
model. At times, the 3-D animation did point out to students their errors in the code, such as incorrect force directions or initial velocity directions. However, more research needs to be done on the extent to which the 3-D animation affects student reasoning, and how instructors can help students make the best use of the resource of 3-D modeling. This becomes especially important in the second semester of the M&I course on electromagnetic phenomena, where VPython is used for the visualization of vector fields in 3-D. VPython is also capable of displaying graphical plots of data in real time in addition to 3-D animations; for example, a 3-D model of a planet orbiting a star can be accompanied by a plot of the kinetic and potential energies as a function of time or of separation. Further study needs to be done on how well students can interpret information from these graphs, and how well they can make connections and translations between the information from the graphs and the behavior of the 3-D model.
6 References cited


7 Appendix
7.1 Appendix 1: Instructions and problems for "hard problems" pilot study

[Pre-interview briefing]
I'm going to present to you a series of physics problems. Some of these problems you might be able to solve. Others you may not be able to solve completely, but you might have some ideas as to how they could be solved, even if you do not know enough to carry out the computations. For the following problems, I would like you to write the steps you would take to obtain a solution, including any relevant equations. **Write out the solution as if you were trying to get partial credit on an exam.** When outlining a solution, you can imagine that you have any resource available such as your textbook (which is available for you to use), a graphing calculator, or a computer. I may ask follow-up questions about your approach to these problems.

When working on the problems, write out as much of your approach as you can on the whiteboards. Try not to erase unless you make an obvious mistake. You may use as many boards as you want.

Since I am interested in what comes to mind as you work on these problems, I'll ask you to TALK ALOUD as you work on the problems. That is, I want you to say out loud EVERYTHING that you are thinking from the time you first see the question until you are done with the problem. Don't plan out what you are saying. Just act as if you are alone in the room speaking to yourself. If you are silent for any length of time, I'll remind you to keep talking aloud. Do you understand what I want you to do?
1. You kick a ball, mass 0.2 kg, from close to ground level, at an angle of 38 deg. above ground, at a speed of 12 m/s.
   - How would you predict where the ball would be 0.90 s from time you kick it?
2. As before, you kick a ball, mass 0.2 kg, from close to ground level, at an angle of 38 deg. above ground, at a speed of 12 m/s.

In the previous problem, air resistance was ignored. In fact, the force due to air resistance acts on the ball while it is moving. The formula for the air resistance force is \( F = kv^2 \), where \( v \) is the speed of the ball, and \( k \) is a constant that depends on the density of the air and the shape of the object. The direction of the force is always in the opposite direction of the ball's velocity. Suppose \( k = 9 \times 10^{-4} \text{ N/(m/s)}^2 \) in this example.

The components of the air resistance force are given by:

\[
F_x = -kv_x \sqrt{v_x^2 + v_y^2}, \quad F_y = -kv_y \sqrt{v_x^2 + v_y^2}
\]

- Now, how would you predict the position of the ball after 0.90s?
3. A satellite is in a circular orbit around the earth at a radius of $4 \times 10^7$ m from the center of the earth. The earth has a mass of $6 \times 10^{24}$ kg. The satellite has a mass of 1000 kg.

   • How would you predict where the satellite would be in 15 hours?
4. As before, a satellite is in a circular orbit around the earth at a radius of $4 \times 10^7$ m from the center of the earth. The earth has a mass of $6 \times 10^{24}$ kg. The satellite has a mass of 1000 kg.

The satellite reaches the starting point again. At this point, the satellite, which has a small rocket engine on board, fires its rockets in the direction of motion. The rocket applies a 400 N force in the direction of the satellite velocity for 10 minutes.

- Now, after the rocket fires, how could you predict where will the satellite be in 15 hours?
5. A block sits on flat, horizontal table and is attached to a spring. The other end of the spring is attached to a vertical post coming out of the table, and the spring is free to pivot around the post. The frictional force between the block and the table is negligible. The block has a mass of 0.20 kg, and the spring constant is 5 N/m. The length of the spring, when relaxed, is 0.16 m.

As shown in the diagram, at a particular instant, the block is moving 0.35 m/s at an angle of 8 degrees below the +x direction. The spring is stretched 0.04 m, and makes an angle of 15 degrees with the –y direction as shown.

- How would you predict where the block will be a time 6 s from this instant?
Appendix 2: Instructions and problems for experiment 1 (main study)

[Pre-interview briefing]
I'm going to present to you a series of physics problems. I would like you to try to solve these problems to the best of your ability. These problems are difficult. That is because I am interested in how students approach hard problems. Don't feel bad if you are unable to solve them completely. If you are not sure how to solve a problem, write out as much of the solution as you can, as if you were trying to get partial credit on an exam.

Your textbook is provided here for your use. Feel free to refer to it if you need it when solving the problems.

You are also allowed to use technology to help in your solution. There are standard and graphing calculators available. Also, if you wish to use a computer to solve any of these problems, there is one available for you to use.

When working on the problems, write out as much of your approach as you can on the whiteboards. Try not to erase unless you make an obvious mistake. You may use as many boards as you want.

Since I am interested in what comes to mind as you work on these problems, I'll ask you to talk aloud as you work on the problems. That is, I want you to say out loud everything that you are thinking from the time you first see the question until you are done with the problem. Don't plan out what you are saying. Just act as if you are alone in the room speaking to yourself. If you are silent for any length of time, I'll remind you to keep talking aloud. Do you understand what I want you to do?

The first problem is a warm-up. It is intended just to get you used to talking aloud and writing on the whiteboards.

[Practice task]
Starting at a traffic light, a car moves North at 13.5 m/s for 110 s. It then makes a sharp right turn (without stopping) and moves east at 11.3 m/s for 70 s. It then speeds up to 12 m/s and travels for another 80 s, at which point it enters a tunnel.

Predict the magnitude of the displacement of the car from the traffic light to the tunnel entrance.

[On the following pages are the problem statements that were provided to the students, as well as examples of VPython programs that solve the problems and their graphical output. No VPython code, example programs, or pictures of VPython graphical displays were provided to students.]
1.) A satellite is in orbit around the earth. At a point where it is $4 \times 10^7$ m from the center of the earth, it is traveling with a speed of $3.0 \times 10^3$ m/s in the direction shown on the diagram. At this point, the satellite, which has a small rocket engine on board, fires its rocket in the direction of motion. The rocket applies a 400 N force in the direction of the satellite's velocity for 10 minutes. The earth has a mass of $6 \times 10^{24}$ kg. The satellite has a mass of 1000 kg.

Predict where the satellite will be in 15 hours.
Carter's program for problem 1. Martin wrote a similar program, but made a few minor mistakes.

```python
from visual import *
sat=sphere(pos=(0,4e7,0), color=color.blue, radius=4e6)
sat.v=vector(2897.8, 776.5, 0)
sat.mass= 1000
Earth=sphere(pos=(0,0,0), color=color.green, radius=6e6)
Earth.mass = 6e24
G = 6.7e-11
dt=100
sat.p = sat.mass*sat.v

while t<54000:
    rate(50)
    Res=Earth.pos-sat.pos
    Force= G*(Earth.mass*sat.mass)/ mag(Res)**2 * norm(Res)
    if t<=600:
        Force=Force+ 400*norm(sat.p)
    sat.p=sat.p+Force*dt
    sat.pos = sat.pos + sat.p/sat.mass * dt
    trail.append(pos=sat.pos)
    t=t+dt

print "Satellite's position", sat.pos
```

Figure 7.1. Comparison of the satellite's trajectory with and without the 400 N force for 10 minutes in problem 1. The above program displays the "with rocket" trajectory. The trail has been added for clarity.
[Instructions for problem 2]

2.) You kick a ball with a mass of 0.5 kg from close to ground level, at an angle of 38 deg. above ground, at a speed of 12 m/s.

Air resistance acts on the ball while it is moving. The formula for the air resistance force is $F = kv^2$, where $v$ is the speed of the ball, and $k$ is a constant that depends on the density of the air and the shape of the object. The direction of the force is always in the opposite direction of the ball's velocity. In this case, $k = 9 \times 10^{-4} \text{ N/(m/s)}^2$.

- Predict the position of the ball after 0.90s

**Possibly helpful information:** the components of the air resistance force are given by

$$F_x = -kv_x\sqrt{v_x^2 + v_y^2}, \quad F_y = -kv_y\sqrt{v_x^2 + v_y^2}$$
[Carter's program for problem 2.]

```python
from visual import *

ball=sphere(pos=(0,0,0), color =color.blue, radius=.5)
baby.v=vector(9.46, 7.39, 0)
baby.mass= .5
earth = box(pos=(4,0,0), length= 9, height =.1)
g=vector(0,-9.81,0)
k=9e-4

dt=.009
ball.p = ball.mass*ball.v
t=0
while t<.9:
    rate(50)
    Fg= ball.mass*g
    Fair= -k*mag(ball.p/ball.mass)**2 * norm(ball.p)
    Fnet= Fair+Fg
    ball.p=ball.p+Fnet*dt
    ball.pos = ball.pos + ball.p/ball.mass * dt
    trail.append(pos=ball.pos)
    t=t+dt
print "Ball's position", ball.pos
```

Figure 7.2. Graphical output of the above program, with the addition of a trail. The position of the ball in the case of negligible air resistance differs by only a few centimeters.
3.) A block sits on a flat, horizontal table and is attached to a spring. The other end of the spring is attached to a vertical post coming out of the table, and the spring is free to pivot around the post. The frictional force between the block and the table is negligible. The block has a mass of 0.20 kg, and the spring constant is 5 N/m. The length of the spring, when relaxed, is 0.16 m.

As shown in the diagram, at a particular instant, the block is moving 0.35 m/s at an angle of 40 degrees below the x-axis. The spring is stretched 0.04 m, and makes an angle of 15 degrees below the x-axis as shown.

• Predict where the block will be at a time 6 s from this instant
[Martin's program for problem 3.]

from visual import *
mBlock=.2
k=5
li=.16
vi=.35
si=.04

block=sphere(pos=vector((li+si)*cos(-.2618), (li+si)*sin(-.2618), 0),
    momentum=mBlock*vector(vi*cos(-.6981), vi*sin(-.6981), 0),
    radius=.05, color=color.blue)
peg=arrow(axis=(0,0,.2), color=color.red)
time=0
dt= 6.0 / 300000

while(time < 6):
    r=(mag(block.pos) - li) * norm(block.pos)
    force=-(k*r)

    block.momentum=block.momentum+force*dt
    block.pos=block.pos + (block.momentum / mBlock) * dt
    time=time+dt

print block.x
print block.y

Figure 7.3. Graphical output of the above program, with the addition of a trail. The sphere represents the block on the spring that is attached to the peg at the origin.
Appendix 3: Instructions and problems for experiment 2

[Briefing before the start of session 1]

In this session and in the two to follow, I'm going to ask you to work on VPython programs similar to the ones you will do (or have done) for physics class. I'm interested in what aspects of programming students find easy, difficult, or confusing. By watching you while you work, I can gain more insight into these issues than from simply examining a finished program you turn for class.

I understand that this is new to you and that you're not an expert. That's OK. I don't expect you to get it right the first time, or not to make mistakes. If you get stuck, you can ask for help from me at any time, or you can look at VPython's online help. I will ask you, however, not to look at programs you've previously written.

I'd like you to talk aloud as you're working. That is, I'd like you say out loud everything that comes to mind while you are writing these programs. Don't plan out what you are going to say. Rather, just act as if you are in the room alone speaking to yourself. If you're silent for any length of time, I'll remind you to keep talking.

To get used to doing this, I'd like you to now go to the course web page, talking aloud as you do it.

[After performing this practice task, students were instructed to start the IDLE VPython editor, and were given the problem statement. ]

[On the following pages are the problem statements that were provided to students in Experiment 2. Each problem statement is followed by an example of a program in VPython code that solves the problem, and a still frame of the graphical output that the running program displays. NO programs or figures were provided to students in the study. The examples are similar to students' completed programs.]
Instructions for program 1, session 1, phase 1 only

For this problem, you'll use VPython cylinder objects to make a 3D picture of a dining room consisting of a circular table and stools.

Open VPython by double clicking the "IDLE for VPython" icon on the desktop. Open the file "shell_1.py" located in the "My Documents" folder. The program shell gives the usual opening lines of a program. It also creates the floor of the dining room. The origin of the coordinate system is above the floor (centered on a hanging light fixture).

(a) Make a cylindrical table. Call it "table". The top of the table is at <0.7, -1.2, 3.1> m. It is 0.8 m high. Its radius is 1.3 m.

(b) Make a stool sitting next to the table. Call it "stool1". The top of the stool is at <3, -1.7, 3.1>. Its radius is 0.25 m, and it is 0.3 m high.

(c) Make an arrow to represent the position vector of the table. Call it "t_a".

(d) Make an arrow to represent the position of the stool. Call it "s_a".

(e) A cup is at the top center of the table. You move it to the top of the stool. Make an arrow to represent the displacement vector of the cup. Call it "ts_a"

(f) A second stool sits next to the table. The vector from the top center of the table to the top of the stool is <-.6, -.5, -2.1> m. What is then the position of the stool with respect to the origin? Make this second stool, with the same radius and height as stool1. Call it "stool2".

(g) Make an arrow that points from the top center of the table to the top of stool2. Call it "ts_a2".

(h) Make an arrow that is the same magnitude and direction as ts_a, but starts at stool2. Call it "newarrow".

(i) Change "newarrow" so that it is in the opposite direction of ts_a and is 3 times as long as ts_a.
Example of a completed program 1, session 1, phase 1 only. Note that the first 13 lines, up to the third row of pound symbols, was provided for students. The first six lines, along with the line that reads "axes(5)" were provided to students for programming assignments in the Spring 2003 M&I course. The purpose of these lines was to draw coordinate axes on the graphical display. They were dropped from the course in Fall 2003, and therefore were not provided to students in phase 2.

The three lines that start with "for i in range(7)" were provided only in this program to draw a gridline "floor" as a reference for ground level.

```python
from visual import *
from __future__ import division
def axes(a=10):
    axes=curve(pos=[(-a,0,0),(a,0,0),(0,0,0),(0,-a,0),(0,a,0),(0,0,0),
                    (0,0,-a),(0,0,a)], color=(0.7,0.7,0.7))
    scene.autoscale = 0
    ########################
    for i in range(7):
        curve(pos=[(2*i-6,-2,-6),(2*i-6,-2,6)], color=color.green)
        curve(pos=[(-6,-2,2*i-6),(6,-2,2*i-6)], color=color.green)
    ########################
    axes(5)
   ########################
    table=cylinder(pos=(0.7,-1.2,3.1), radius=1.3, axis=(0,-.8,0))
    stool1=cylinder(pos=(3, -1.7, 3.1), radius=.25, axis=(0,-.3,0))
    t_a=arrow(pos=(0,0,0),axis=(table.pos), color=color.green)
    s_a=arrow(pos=(0,0,0),axis=(stool1.pos), color=color.yellow)
    ts_a=arrow(pos=(table.pos),axis=(stool1.pos-table.pos),
               color=color.magenta)
    stool2=cylinder(pos=(table.pos+vector(-.6,-.5,-2.1)),
                    radius=stool1.radius, axis=stool1.axis)
    ts_a2=arrow(pos=(table.pos), axis=(stool2.pos-table.pos),
                color=color.blue)
    newarrow=arrow(pos=(stool2.pos), axis=(-3*ts_a.axis),
                color=color.green)
```
Figure 7.4. Sample graphical output from program 1, session 1, phase 1 only (table and stools).
[Instructions for program 1, session 1, phase 2 only]

In this program you'll create a (static) model of the planet Jupiter and three of its moons: Io, Europa, and Ganymede.

(a) Place Jupiter at the origin. Jupiter has a radius of $7.15 \times 10^7$ m.

(b) Place Io on the positive y axis, a distance of $4.22 \times 10^8$ m from the center of Jupiter. Give it a radius of $1.82 \times 10^7$ m. (The radii of the three moons in this program are actually 10 times larger than their real values.)

(c) Place Europa on the positive x axis, a distance of $6.71 \times 10^8$ m from the center of Jupiter. Give it a radius of $1.57 \times 10^7$ m.

(d) Place Ganymede on the negative y axis, a distance of $1.07 \times 10^9$ m from the center of Jupiter. Give it a radius of $2.63 \times 10^7$ m.

(e) Create an arrow that represents the position vector of Ganymede (with respect to Jupiter). Call this arrow "ga".

(f) Create an arrow that represents the relative position of Europa with respect to Io. Call it "e_i".

(g) Imagine a spacecraft is traveling from Ganymede to Europa, and that it is currently halfway between Ganymede and Europa. Create an arrow that represents the current position of the spacecraft with respect to Ganymede. Call this arrow "craft_arr".

(h) Create an arrow that has the same magnitude and direction as "craft_arr," but starts at Io. Call it "newarrow".

(i) Change "newarrow" so that it is in the opposite direction as "craft_arr," and is three times as long.
Example of a completed program 1, session 1, phase 2 only

from visual import *
Jupiter = sphere(pos=(0,0,0), radius=7.15e7, color=color.orange)
Io = sphere(pos=(0,4.22e8,0), radius=1.82e7)
Europa = sphere(pos=(6.71e8,0,0), radius=1.57e7, color=color.yellow)
Ganymede = sphere(pos=(0,-1.07e9,0), radius=2.63e7, color=color.green)
ga = arrow(pos=(0,0,0), axis=(0,-1.07e9,0), color=color.magenta)
e_i = arrow(pos=(0,4.22e8,0), axis=(6.71e8,-4.22e8,0), color=color.cyan)
craft_arr = arrow(pos=(0,1.07e9,0), axis=(6.71e8/2,1.07e9/2,0),
                 color=color.red)
newarrow = arrow(pos=(0,4.22e8,0), axis=(-3*6.71e8/2,-3*1.07e9/2,0),
                  color=color.blue)

Figure 7.5. Sample graphical output from program 1, session 1, phase 2 only (static Jovian moons)
[Instructions for program 2, session 1, both phases]

(a) Create a sphere to represent a rock in outer space. Call it "rock". It has a radius of 10 m and is currently at position <-50, -70, 80> m.

(b) The rock has a velocity of <4, 2, -7> m/s. Make the rock move at this velocity for 40 seconds. Show this motion in 10 steps.

(c) Slow down the animation so you can see the rock move

(d) Show the rock's path with a trail.

(e) Show the rock's velocity with an arrow attached to the rock.

[Note that some students did not reach parts d and e. The verbal protocols were therefore coded, analyzed, and compared only up to the point where students completed part c.]
Example of a completed program 2, session 1, up to and including part c. Students in phase 1 were provided introductory lines to the program that drew coordinate axes. Students in phase 2 were instead instructed to place a sphere at the origin to serve as a reference point, called "A" in the program below.

```python
from visual import *

A      = sphere(pos=(0, 0, 0), radius=10)
rock   = sphere(pos=(-50, -70, 80), radius=10, color=color.blue)
rock.m = 4e6
rock.p = rock.m*vector(4, 2, -7)
deltat = 4
t = 0
while t<=40:
    rate(5)
    rock.pos = rock.pos + (rock.p/rock.m)*deltat
    t = t + deltat
```

Figure 7.6. Sample graphical output for program 2, session 1 (moving rock). The rock (dark sphere) is the same size as the lighter sphere representing the origin, but has moved past the origin in the –z direction.
a) Write a program to model the moon's orbit around the earth. The moon's mass is $7.4 \times 10^{22}$ kg, and the earth's mass is $6.0 \times 10^{24}$ kg. Set the earth to be at the origin, and set the moon's initial position to be $<3.8 \times 10^8, 0, 0>$ m.

The earth's radius is $6.4 \times 10^6$ m. The moon's radius is $1.75 \times 10^6$ m. You may need to make the radii larger than the real values to be able to clearly see the earth and moon in the graphics window.

The moon's orbit is nearly circular, and it makes a complete revolution once every 28 days. Use this when deciding on the moon's initial momentum.

Since the earth is about 100 times more massive than the moon, assume that the earth remains stationary. Ignore the earth's orbit around the sun.

b) Make the moon leave a trail. Is the trail closed? What should be done to make it close?

c) Make arrows to represent the moon’s momentum, force on moon by the earth.

d) Change the initial velocity of the moon so that you get an elliptical orbit. If you need to, change delta t so that the orbit is closed once again.

e) What would happen if the gravitational force law were something different? Try changing the force law so that it depends on the inverse cube of distance instead of the inverse square.

f) Change the force law back to what it was originally. Now, update the momentum and position of the earth as well as the moon.

What do you notice about the motion now?

g) What do you have to do to keep the earth/moon system stationary, and not to “wander off”?

h) Change the mass of the moon to 10 times the original mass. How do the orbits of the earth and moon change?
from visual import *

earth=sphere(pos=(0,0,0),radius=10*6.4e6,color=color.red)
moon=sphere(pos=(3.8e8,0,0),radius=10*1.75e6,color=color.blue)
earth.m=6e24
moon.m=7.4e22
gravitation=6.67e-11
t=0
moon.p=moon.m*vector(0,986.4,0)
deltat=3600

while t<28*24*3600:
    t=t+deltat
    rate(100)
    r=earth.pos-moon.pos
    rmag=sqrt(r.x**2+r.y**2+r.z**2)
    rhat=r/rmag
    Fnet=(gravitation*earth.m*moon.m*rhat)/rmag**2
    moon.p=moon.p+Fnet*deltat
    moon.pos=moon.pos+(moon.p*deltat/moon.m)

Figure 7.7. Sample graphical output of program 3 (moon orbit). A trail has been added for clarity.
In one window is the skeleton of a VPython program. It has only the introductory line at the beginning of each program, and a while statement. In the other window is a list of sections of lines from a VPython program that models the motion of a planet orbiting a star. The lines are currently arranged in random order. Copy and paste these lines into the program in the correct order, so that the program runs correctly.

[The "introductory line" was the statement "from visual import *" which begins every VPython program. For the randomly arranged program lines, see figure 4.16. For the program skeleton, see figure 4.17.]
Appendix 4: Classroom vector activity, Spring 2003

Classroom Vector Activity

Physical phenomena occur in a 3D world. We need a common language and notation for describing positions in 3D. Today we'll develop that common language and notation.

Notice the white sphere mounted on a tripod. It is about 2.5 m above the floor.

We will use an xyz coordinate system and <x,y,z> notation for recording positions. Use +x for right, +y for up, +z for outward to match VPython coordinates. (We'll discuss VPython later).

1) Each group (of three) should measure the position of the center of their table using the white sphere as the origin. Measure to the nearest tenth-meter (0.1 m or 10 cm); high accuracy is not needed.

2) Your groups at the table should agree on their answer and write the <x,y,z> position on the wallboard.

Discussion: Do all tables report the same y? Why or why not? What is the significance of the sign of y? What about the sign of z? Of x?

Notation

3) If you are at table 4, write the position vector for your table as \( \vec{r}_4 = <x, y, z> \).

The notation \( \vec{r} \) means the "vector" location of table 4. The arrow over the \( r \) means "3 numbers are involved", and the subscript 4 labels the table. The individual numbers are called "components" of the vector.

Question: How would the components (x, y, and z) change if the white sphere moved in the x direction by 1 meter and in the y direction (up) by 1 meter?

4) Pair up by tables: 1-11, 2-4, 3-10, 5-9, 6-8 (table 4 use positions of 4 and 8). Calculate \( <x_2-x_1, y_2-y_1, z_2-z_1> \) etc.; Report the difference between your table and the other table in your pair. For example, if your table is 3 and the other is 4, report \( \vec{r}_3 - \vec{r}_4 = <x,y,z> \) on the board.

5) What is the physical significance of this vector subtraction? What quantity does it represent? Write your statement on the board.

Discussion: What is the significance of the sign flip for pairs of tables?

Question: How would the numbers change if the white sphere moved in the x direction by 1 meter and in the y direction (up) by 1 meter?
6) Next, each group should measure directly the vector from the center of the paired table to the center of your own table. Using the coordinate system for the class, report this vector that describes the displacement of one table from another, using \( <x, y, z> \). Does this vector agree with your answer obtained by subtracting the original vectors?

**Magnitude**

The magnitude of a 3D vector is \( \sqrt{x^2 + y^2 + z^2} \) (3D version of the Pythagorean theorem).

7) Calculate the magnitude between your table and your paired table, using this magnitude formula, and write it on the board in the form \( \sqrt{x^2 + y^2 + z^2} = \text{your result} \).

Notation: \( |\vec{r}_4| \) (or \( r_4 \) without the vector sign above the \( r \)) means magnitude.

8) Next, measure the actual distance between adjacent table centers, and write it on the board in the form \( |\vec{r}_4| = \text{your result} \). Does it agree with your answer from using the formula \( \sqrt{x^2 + y^2 + z^2} \)?

Question: What would happen to the magnitude if we chose a different origin (location for the white sphere) or the axes were pointing in very different directions (but still perpendicular to one another)?

How does magnitude \( |\vec{r}_4| \) contrast with vector \( \vec{r}_4 = <x, y, z> \)?

**Unit vector**

A unit vector has magnitude 1. Unit vectors are useful in specifying a direction. A unit vector is written with a "hat" over it; the unit vector \( \vec{r} \) is called "r-hat". You can construct a unit vector by dividing a vector by its magnitude:

\[ \hat{r} = \frac{\vec{r}}{r} \]

9) Construct a unit vector pointing from your table to your paired table; write its value on the board in the form \( \hat{r} = \frac{\vec{r}}{r} = <x, y, z> \).

We will make frequent use of unit vectors in our calculations.

[This activity was accompanied by a worksheet. See the "Vector Minilab" on the following pages.]
Mini-lab: 3D vectors

Table number \( n = \) _______. Print your name: ____________________________

Print names of partners who worked on this mini-lab:

Position vector to your table number \( n \): \( \vec{r}_n = \langle \quad \quad, \quad \quad, \quad \quad \rangle \) meters

Magnitude of position vector \( |\vec{r}_n| = r_n = \) _______ meters

Show what you entered into your calculator for calculating the magnitude:

Other, neighboring table number \( m = \) _______.

Position vector to the other table, table number \( m \): \( \vec{r}_m = \langle \quad \quad, \quad \quad, \quad \quad \rangle \) meters

Magnitude of position vector \( |\vec{r}_m| = r_m = \) _______ meters

Show what you entered into your calculator for calculating the magnitude:

Position of your table \( n \) relative to other table \( m \) (a vector pointing from table \( m \) to table \( n \)):

\[ \vec{r}_{nm} = \vec{r}_n - \vec{r}_m = \langle \quad \quad, \quad \quad, \quad \quad \rangle \) meters

Magnitude of relative position vector: \( |\vec{r}_{nm}| = r_{nm} = \) _______ meters

Show what you entered into your calculator for calculating the magnitude:

Calculate \( |\vec{r}_n| - |\vec{r}_m| = \) _______ meters. Is \( |\vec{r}_n - \vec{r}_m| = |\vec{r}_n| - |\vec{r}_m| \) ? _______

Position of other table \( m \) relative to your table \( n \) (a vector pointing from table \( n \) to table \( m \)):

\[ \vec{r}_{mn} = \vec{r}_m - \vec{r}_n = \langle \quad \quad, \quad \quad, \quad \quad \rangle \) meters

Magnitude of relative position vector: \( |\vec{r}_{mn}| = r_{mn} = \) _______ meters

Calculate unit vectors, showing your calculations:

\[ \hat{\vec{r}}_n = \frac{\vec{r}_n}{|\vec{r}_n|} = \]

\[ \hat{\vec{r}}_m = \frac{\vec{r}_m}{|\vec{r}_m|} = \]

\[ \hat{\vec{r}}_{nm} = \frac{\vec{r}_{nm}}{|\vec{r}_{nm}|} = \]

\[ \hat{\vec{r}}_{mn} = \frac{\vec{r}_{mn}}{|\vec{r}_{mn}|} = \]

(Turn to other side)
An object’s location is described by a position vector $\mathbf{r}_a = (-2, 3, 0)$ meters. On the diagram above, draw the vector $\mathbf{r}_a$ and label it.

A second object’s location is described by a position vector $\mathbf{r}_b = (3, 5, 0)$ meters. On the diagram above, draw the vector $\mathbf{r}_b$ and label it.

On the diagram above, draw the vector $\mathbf{r}_{ba} = \mathbf{r}_a - \mathbf{r}_b$ and label it. Put the tail of the vector at the location of the second object.

Check that your name and the names of your partners are on the front page.
7.5 Appendix 5: VPython Introduction, Spring 2003

VPython Introduction for PY 205

1 Overview

VPython is a programming language that is easy to learn and is well suited to creating 3D interactive models of physical systems. VPython has three components that you will deal with directly:

- Python, a programming language invented in 1990 by Guido van Rossum, a Dutch computer scientist. Python is a modern, object-oriented language which is easy to learn.
- Visual, a 3D graphics module for Python created by David Scherer while he was a student at Carnegie Mellon University. Visual allows you to create and animate 3D objects, and to navigate around in a 3D scene by rotating and zooming, using the mouse.
- IDLE, an interactive editing environment, written by van Rossum and modified by Scherer, which allows you to enter computer code, try your program, and get information about your program. IDLE is currently not available on pre-OSX Macintosh.

This tutorial assumes that Python and Visual are installed on a Windows computer that you are using. You can obtain VPython at no charge from http://vpython.org, which also has instructions for use on Linux or Macintosh.

2 Your First Program

> Start IDLE by double-clicking on the IDLE for VPython icon in the Math & Statistics section of the Unity application launcher (or the icon on the desktop of your own computer). A window labeled “Untitled” should appear. This is the window in which you will type your program.

> Go to the course web site and choose the link marked Software, and from there choose VPython Commands. From there choose "Lines to insert at the beginning of your program". Copy these lines, and insert them at the beginning of your file.

These statements instruct Python to use the Visual graphics module, and to draw axes.

> After the lines you inserted in your program, type the following statement:

```
cylinder()
```

Your program should look like this:

```
from visual import *
from __future__ import division
def axes(a=10):
  axes=curve(pos=[(-a,0,0),(a,0,0),(0,0,0),(0,-a,0),(0,a,0),(0,0,0),
                  (0,0,-a),(0,0,a)], color=(0.7,0.7,0.7))
scene.autoscale = 0
########################################################################
axes(10)  # if necessary, change "10" to an appropriate number for your program

cylinder()
```

3 Running the Program

> Now run your program by pressing F5 (or by choosing “Run program” from the “Run” menu).

When you run the program, two new windows appear. There is a window titled “VPython,” in which you should see white axes and a white cylinder, and a window titled “Output.” Move the Output window to the bottom of your screen where it is out of the way but you can still see it (you may make it smaller if you wish).

> In the VPython window, hold down both mouse buttons and move the mouse (this may take two hands on a laptop). You should see that you are able to zoom into and out of the scene.

> Now try holding down the right mouse button. You should find that you are able to rotate around the scene.

To avoid confusion about motion in the scene, it is helpful to keep in mind the idea that you are moving a camera around the object. When you zoom and rotate you are moving the camera, not the objects.
4 Stopping the Program

Click the close box in the upper right of the display window (VPython window) to stop the program. Leave the Output window open, because this is where you may receive error messages.

5 Save Your Program

Save your program by pulling down the File menu and choosing Save As. Give the program a name ending in " .py," such as "MyProgram.py". You must type the " .py" extension; IDLE will not supply it. Every time you run your program, IDLE will save your code before running. You can undo changes to the program by pressing CTRL-z.

6 Modifying Your Program

Let’s change the position, orientation, and size of the cylinder to represent approximately the instructor’s table. (By default, the position is at the origin, and the axis of the cylinder points along the x axis.)

> Change the last line of your program to read:

\[
cylinder(\text{pos}=(-0.25, -1.5, 0), \text{axis}=(0, -1, 0), \text{radius}=0.3)
\]

What does this line of code do? To position objects in the display window we set their 3D cartesian coordinates. The origin of the coordinate system is at the center of the display window. The positive x axis runs to the right, the positive y axis runs up, and the positive z axis comes out of the screen, toward you. The assignment

\[
\text{pos}=(-0.25, -1.5, 0)
\]

sets the position of the cylinder by assigning values to the x, y, and z coordinates of the center of one end of the cylinder. Here we have chosen the position to be the top of the instructor’s table. The assignment

\[
\text{axis}=(0, -1, 0)
\]

makes the axis of the cylinder point downward, with a length of 1. Finally,

\[
\text{radius}=0.3
\]

gives the cylinder a radius of 0.3 in the same units as those used for the position and axis.

We can also give the cylinder a distinctive color. The following statement

\[
cylinder(\text{pos}=(-0.25, -1.5, 0), \text{axis}=(0, -1, 0), \text{radius}=0.3, \text{color}=\text{color.red})
\]

makes the cylinder red (there are 8 colors easily accessible by \text{color.xxx}: red, green, blue, yellow, magenta, cyan, black, and white).

> Now press F5 to run your program.

The cylinder created by your program is an object. Properties like pos, axis, radius, and color are called attributes of the object.

Give the cylinder representing the instructor’s table a name “table0” so that we can refer to it later if necessary:

\[
\text{table0} = \text{cylinder(\text{pos}=(-0.25, -1.5, 0), \text{axis}=(0, -1, 0), \text{radius}=0.3, \text{color}=\text{color.red})}
\]

> Using the classroom data obtained earlier, create a second cylinder to represent table 1. Give it the name “table1”. Give it any color you like.

7 The vector object

> At the end of your program add the following statement

\[
x_0=\text{vector}(-0.25, -1.5, 0)
\]

which creates a vector object, named \text{x}_0, which represents mathematically the position vector of the center of the top of the instructor’s table. Run the program (press F5), and you don’t see anything new in the scene. The vector object doesn’t create a visible object. Rather it creates a vector to be used for computational purposes.

> Add the statement

\[
\text{print } "\text{Position of instructor table ="}, x_0
\]
Run the program, and look in the Output window, where you will see the components of the vector displayed.

> Here is a way to calculate and print the magnitude of the vector. Add this to your program and run it:

\[
\text{x}_0\text{mag} = \sqrt{\text{x}_0.x^2 + \text{x}_0.y^2 + \text{x}_0.z^2}
\]

\[
\text{print \"Magnitude of } \text{x}_0\text{ \"}, \text{x}_0\text{mag}
\]

There are three new concepts here. The function sqrt calculates the square root. The individual components of the VPython vector \( \text{x} \) are \( \text{x}.x, \text{x}.y, \) and \( \text{x}.z \). The form \( \text{a}**\text{b} \) means “raise \text{a} to the power of \text{b}” (exponentiation).

8 Referring to attributes of objects

You saw that the components of vector \( \text{x}_0 \) are \( \text{x}_0.x, \text{x}_0.y, \) and \( \text{x}_0.z \). The pos and axis attributes of a cylinder object are also vectors.

> Add this to your program:

\[
\text{print \"Position of table 1 \"}, \text{table1.pos}
\]

\[
\text{print \"Axis of table 1 \"}, \text{table1.axis}
\]

You should see the components of the vectors representing the position and axis of table 1. Similarly, you can use \( \text{table1.pos} \) anywhere in vector calculations.

9 The arrow object

The vector object creates an object that you can use in computations, but it doesn’t display an arrow on the screen, which is the purpose of the arrow object.

> Display an arrow pointing from the origin to the top of the instructor’s table:

\[
\text{ar}_0 = \text{arrow(pos}(0,0,0), \text{axis}=\text{x}_0, \text{shaftwidth}=0.2, \text{color}=\text{color.green})
\]

You should see a green arrow pointing to the center of the top of the instructor’s table. The arrow object gives you a visualization of a vector.

10 Writing the 3D Classroom Model

Your first VPython assignment is to write a program that is a 3D model of the classroom. You can start with the program you have written above. You can find a detailed description of the assignment on the course web site, on the Calendar for the first or second week. If you go to the Software section of the course website, you will find a list of VPython commands, which contains descriptions of the vector, cylinder, and arrow objects, which you will need for this program.
7.6 Appendix 6: 3-D classroom model computer activity, Spring 2003

3D Classroom Model (In-class computer activity)

Using the data from the vector minilab you did in class, you will now write a computer program in which you:

Create and display a 3D model of the classroom

Create and display 3D arrows to represent the position vectors you calculated

You will turn in the program via WebAssign, by uploading the file.

Your instructor will demonstrate how to open IDLE, the VPython editor, and how to create your first VPython program. Also see VPython documentation under Software on the course website.

Data you will need before you begin: (in S.I. units, as usual)

The coordinates of the center of each table (there are 11 tables)

The distance from the center of a table to the floor

The radius of a table

What should be in the program:

1) Lines for origin and axes to insert at the beginning of your program

Copy these lines and paste them at the beginning of your program.

```python
from visual import *
from __future__ import division
def axes(a=10):
    axes=curve(pos=[(-a,0,0),(a,0,0),(0,0,0),(0,-a,0),(0,a,0),(0,0,0),
                      (0,0,-a),(0,0,a)], color=(0.7,0.7,0.7))
    scene.autoscale=0

axes(10)    # if necessary, change "10" to an appropriate number for your program
```

2) Represent each table by a cylinder whose radius is the radius of a table. The "position" (pos) of the cylinder should be the (x,y,z) coordinates of the center of the table, and the "axis" of the cylinder should be a vector pointing from the center of the table straight down to the floor.
3) Create a vector variable named $r_n$ (where "n" is the number of your table, for example $r_6$), pointing from the origin to the center of your table. Print the value of $r_n$.
Create a scalar variable named $r_n_{mag}$, and set it equal to the magnitude of $r_n$ (which you must calculate).
Print the value of $r_n_{mag}$.

4) Create an arrow named $ra_n$ (where "n" is the number of your table) to visualize the vector $r_n$.
Put the tail of the arrow (pos) at the origin.
Set the axis of the arrow equal to your vector $r_n$.
Set the shaftwidth of the vector equal to 0.2.
Make the arrow green.

5) Create a vector variable named $r_m$ (where "m" is the number of a different table, for example $r_3$), pointing from the origin to the center of table "m".
Print the value of $r_m$.

6) As in part 4), create an arrow named $ra_m$ to visualize the vector $r_m$.
Make the arrow yellow.

7) Create a vector variable named $r_{m\_n}$, and set it equal to the vector difference $r_n - r_m$.
Print the value of the vector $r_{m\_n}$.
Create a scalar variable named $r_{m\_n\_mag}$, and set it equal to the magnitude (which you must calculate) of $r_{m\_n}$.

8) Create an arrow named $ra_{m\_n}$, to visualize the vector $r_{m\_n}$.
Make the arrow red.
The arrow should point from table $m$ to table $n$.

* * *

**VPython commands you should now be able to use:**
cylinder
arrow
vector
print

You should also be able to calculate squares and square roots ($x**2$ and sqrt($x$))
7.7 Appendix 7: 3-D Model of Motion activity, Spring 2003

[In week 2 of the Spring 2003 semester, students were introduced to the concept of velocity. Following this, the instructor demonstrated how to write a program that modeled the motion of a ball moving at constant velocity. In the SCALE-UP classroom, the instructor could display on the projector what he typed into the computer as he was typing. Students, who were seated at laptop computers either individually or in groups of two, typed exactly what the instructor wrote into their own programs. This program met the following instructions.]

3D Model of Motion (In-class computer activity)

Write a VPython program to model the motion of a ball moving at constant velocity. The program you submit in WebAssign should include the following:

- Copy the usual lines and insert them at the beginning of your program.
- Create a ball of radius 0.5 m, initially at location <-10,-5,0> m.
- Give the ball a constant velocity so that it moves to the right (+x) 1.0 m and toward you (+z) 0.5 m every second.
- Use a time step of 2 seconds.
- Inside a while loop, update the ball's position for 10 time steps.
- After each time step, print the total elapsed time, and the (vector) position of the ball.
- Slow the program down with a rate(5) command inside the loop.
- Make the ball leave a trail showing its path.
- Display an arrow at the (moving) location of the ball, pointing in the direction of the ball's velocity. The arrow should move with the ball. Make the arrow long enough to see clearly.
One week after this activity, the instructor again led the students in a computer activity to modify this program so that the object moves under the influence of a constant force. The instructor showed the students how to define an initial momentum, and then asked the students to insert these lines at the start of the "while" loop of the program:

\[
F = \text{vector}(0, 0.01, 0) \\
deltap = F \cdot \text{deltat} \\
\text{ball.p} = \text{ball.p} + \text{deltap} \\
\text{ball.v} = \text{ball.p} / \text{ball.m}
\]

After some discussion about what was happening in the program, the instructor led the students in changing various parameters of the program, including velocity, force, and time step. The instructor also asked the students to change the force to a non-constant expression that depended on the position of the ball:

\[
F = \text{vector}(0, -0.01 \cdot \text{ball.pos.y}, 0)
\]

The implications of this force and the behavior of the program were then discussed.
(1.) What are the components of the vector $\vec{a}$ in Figure 1?

$$\vec{a} = \langle \underline{\quad}, \underline{\quad}, 0 \rangle$$

Note that since the vector lies in the $x$-$y$ plane, the $z$-component is zero.

(2.) What are the components of vector $\vec{b}$ in Figure 2?

$$\vec{b} = \langle \underline{\quad}, \underline{\quad}, \underline{\quad} \rangle$$

(3.) What are the components of vector $\vec{c}$ in Figure 3?

$$\vec{c} = \langle \underline{\quad}, \underline{\quad}, \underline{\quad} \rangle$$
Imagine you have a baseball and a tennis ball at different locations. The coordinates of the center of the baseball are \((3, 5, 0)\) m. The coordinates of the tennis ball’s center are \((-3, -1, 0)\) m. On the graph below, do the following:

a. Draw dots at the coordinates of the baseball and tennis ball.

b. Draw an arrow whose tail is at the origin and whose tip is at the baseball’s center. This is the position vector of the baseball. On the graph, label this position vector \(\vec{B}\). Write the components of vector \(\vec{B}\):

\[
\vec{B} = \langle \underline{\hspace{2cm}}, \underline{\hspace{2cm}}, \underline{\hspace{2cm}} \rangle \text{ m}
\]

c. Draw the tennis ball’s position vector. On the graph, label it \(\vec{T}\). Write the components of vector \(\vec{T}\):

\[
\vec{T} = \langle \underline{\hspace{2cm}}, \underline{\hspace{2cm}}, \underline{\hspace{2cm}} \rangle \text{ m}
\]

d. On the same graph, draw a vector whose tail is at the center of the baseball and whose tip is at the center of the tennis ball. (This is a relative position vector; it gives the position of the tennis ball relative to the baseball.) On the graph, label this vector \(\vec{r}\). Read the components of \(\vec{r}\) and write them below:

\[
\vec{r} = \langle \underline{\hspace{2cm}}, \underline{\hspace{2cm}}, \underline{\hspace{2cm}} \rangle \text{ m}
\]

e. Add the vectors \(\vec{B}\) and \(\vec{r}\) and write the result:

\[
\vec{B} + \vec{r} = \langle \underline{\hspace{2cm}}, \underline{\hspace{2cm}}, \underline{\hspace{2cm}} \rangle \text{ m}
\]

Is this the same as vector \(\vec{T}\)?

f. Perform the subtraction \(\vec{T} - \vec{B}\) and write the result.

\[
\vec{T} - \vec{B} = \langle \underline{\hspace{2cm}}, \underline{\hspace{2cm}}, \underline{\hspace{2cm}} \rangle \text{ m}
\]

Is this the same as vector \(\vec{r}\)?

g. The vector \(\vec{B}\) can be written as an algebraic expression in terms of the vectors \(\vec{r}\) and \(\vec{T}\). Which of the following choices is the correct algebraic expression for \(\vec{B}\)? Circle the correct expression.

(a) \(\vec{B} = \vec{r} + \vec{T}\)  
(b) \(\vec{B} = \vec{r} - \vec{T}\)

(c) \(\vec{B} = \vec{T} - \vec{r}\)  
(d) \(\vec{B} = -\vec{T} - \vec{r}\)
(5.) What is the vector that has its tail at \( \langle 9.5, 7, 0 \rangle \) m and its tip at \( \langle 4, -13, 0 \rangle \) m?

\( \langle \quad , \quad , \quad \rangle \) m

(6.) A man is standing on the roof of a building at the position \( \langle 12, 30, 13 \rangle \) m. He sees the top of a tree, which is at the position \( \langle -25, 35, 43 \rangle \) m. What is the vector that points from the man to the top of the tree?

\( \langle \quad , \quad , \quad \rangle \) m

What is the magnitude of this vector (that is, the distance from the man to the top of the tree)?

(7.) Given a vector \( \vec{a} = \langle 2, 4, 0 \rangle \):

a. Draw vector \( \vec{a} \) on the graph with its tail at the origin.

b. Calculate the vector \( 2\vec{a} \):

\( \langle \quad , \quad , \quad \rangle \).

With its tail at position \( \langle 0, -5, 0 \rangle \), draw vector \( 2\vec{a} \).

c. Calculate the vector \( -0.5\vec{a} \):

\( \langle \quad , \quad , \quad \rangle \).

With its tail at position \( \langle 0, 3, 0 \rangle \), draw vector \( -0.5\vec{a} \).

d. What is the magnitude of vector \( \vec{a} \)?

\( |\vec{a}| = \quad \)

e. What is the magnitude of vector \( 2\vec{a} \)?

\( |2\vec{a}| = \quad \)

f. Is the magnitude of \( 2\vec{a} \) equal to 2 times the magnitude of \( \vec{a} \)? That is, does \( |2\vec{a}| = 2|\vec{a}| \)?

g. Calculate the vector \( \frac{\vec{a}}{|\vec{a}|} \): \( \langle \quad , \quad , \quad \rangle \).

With its tail at position \( \langle -1, 0, 0 \rangle \), draw this new vector.

h. A vector divided by its magnitude is called a “unit vector.” In this case the unit vector \( \hat{a} = \frac{\vec{a}}{|\vec{a}|} \), where \( \hat{a} \) is pronounced “a-hat”. Calculate the magnitude of \( \hat{a} \):
(8.) Calculate the magnitude of \( \vec{q} = (1.5, -2.3, -4.1) \) m: \( \quad \) m

Calculate the unit vector \( \vec{q} : \langle \quad \quad \quad \quad \quad \rangle \)

Now write \( \vec{q} \) as a product of its magnitude and its unit vector: \( \quad \) m \( \langle \quad \quad \quad \quad \quad \rangle \)

(9.) A laser pointer shines a laser beam on a window (see figure below). The laser beam travels through the window (with negligible bending) and hits a building. The distance that the laser beam travels from the beam to the building is 34 m. The relative position vector from the laser pointer to the point where the beam hits the window is \( \vec{r} = (2.1, 0.7, -1.3) \). Calculate the relative position vector from the laser pointer to the point where the beam hits the building.

\( \langle \quad \quad \quad \quad \quad \rangle \) m
Introduction to VPython

This worksheet will guide you through the basics of programming in VPython. Be sure to read the instructions carefully and to do all of the exercises.

VPython is a programming language that allows you to easily make 3-D graphics and animations. We will use it extensively in this course to model physical systems. First we will introduce how to create simple 3-D objects. Then we will use VPython to explore vectors and vector operations in 3-D.

On the screen desktop there should be an icon called “IDLE for Python.” Double click it. This starts IDLE, which is the editing environment for VPython.

1. Starting a program
• Enter the following line of code in the IDLE editor window.

```python
from visual import *
```

Every VPython program begins with this line. This line tells the program to use the 3D module (called “visual”).

Before we write any more, let’s save the program:
• In the IDLE editor, from the “File” menu, select “Save.” Browse to a location where you can save the file, and give it the name “test.py”. YOU MUST TYPE the “.py” file extension --IDLE will NOT automatically add it.

2. Creating a sphere
• Now let’s tell VPython to make a sphere. On the next line, type:

```python
sphere()
```

This line tells the computer to create a sphere object. Run the program by pressing F5 on the keyboard. Two new windows appear in addition to the editing window. One of them is the 3-D graphics window, which now contains a sphere.

3. The 3-D graphics scene
By default the sphere is at the center of the scene, and the “camera” (that is, your point of view) is looking directly at the center.
• Hold down both buttons and move the mouse up and down to make the camera move closer or farther away from the center of the scene.
• Hold down the right mouse button alone and move the mouse to make the camera “revolve” around the scene, while always looking at the center.

When you first run the program, the coordinate system has the positive x direction to the right, the positive y direction pointing up, and the positive z direction coming out of the screen toward you. You can then rotate the camera view to make these axes point in other directions.

4. The text output window
The second new window that opened when you ran the program is the text output window. It is useful to keep this window open, because error messages are displayed here. Also, if you include lines in the program that tell the computer to print text, the text will appear in this window.
• Use the mouse to make the text output window smaller, and move to the lower part of the screen. Keep it open when you are writing and running programs so you can easily spot error messages.
To see an example of an error message, let’s try making a spelling mistake:

- Change the first line of the program to the following:

```python
from bisual import *
```
- Run the program.

Notice you get a message in red text in the output window. The message gives the filename, the line where the error occurred, and a description of the error (in this case “ImportError: No module named bisual”).
- Correct the error in the first line.

5. Attributes

Now let’s give the sphere a different position in space and a radius.
- Change the last line of the program to the following:

```python
sphere(pos=vector(-5,2,-3), radius=0.40, color=color.red)
```
- Run the program.

This line gives the sphere three attributes:
1. `pos`: the position vector of the center of the sphere
2. `radius`: the sphere’s radius
3. `color`: the sphere’s color. Color values are written as “color.xxx”, where xxx could be red, blue, green, cyan, magenta, yellow, black, or white.
- Change the last line to read:

```python
sphere(pos=vector(2,4,0), radius=0.20, color=color.white)
```

Note the changes in the sphere’s position, radius, and color.

Experiment with different values for the attributes of the sphere. Try giving the sphere other position vectors. Try giving it different values for “radius.” Run the program each time you make a change to see the results. When you are done, reset the line to how it appears above (that is, pos=vector(2,4,0), and radius=0.20).

6. Autoscaling and units

VPython automatically “zooms” the camera in or out so that all objects appear in the window. Because of this “autoscaling”, the numbers for the “pos” and “radius” could be in any consistent set of units, like meters, centimeters, inches, etc. For example, this could represent a sphere with a radius 0.20 m and a position vector of <2, 4, 0> m. In this course we will always use SI units in our programs (that is, the system of units based on meters, kilograms, and seconds).

7. Creating a second object

- We can tell the program to create additional objects. Type the following on a new line, then run the program:

```python
sphere(pos=vector(-3,-1,0), radius=0.15, color=color.green)
```

You should now see the original white sphere and a new green sphere. In later exercises, the white sphere will represent a baseball and the green sphere will represent a tennis ball. (The radii are exaggerated for visibility.)

8. Arrows

We often use arrow objects in VPython to depict vector quantities. We next add arrows to our programs.
• Type the following on a new line, then run the program:

```python
arrow(pos=vector(2,-3,0), axis=vector(3,4,0), color=color.white)
```

This line tells VPython to create an arrow with 3 attributes:
1.) `pos`: the position vector of the tail of the arrow. In this case, the tail of the arrow is at the position `<2,-3,0>` m.
2.) `axis`: the components of the arrow vector, that is, the vector measured from the tail to the tip of the arrow. In this case, the arrow vector is `<3,4,0>` m.
3.) `color`: the arrow’s color.

To demonstrate the difference between “pos” and “axis,” let’s make a second arrow with a different “pos” but same “axis.”

• Type the following on a new line, then run the program:

```python
arrow(pos=vector(3,2,0), axis=vector(3,4,0), color=color.red)
```

Note the red arrow starts at a different point than the white arrow, but has the same magnitude and direction. This is because they have the same “axis,” but different values of “pos.”

9. Scaling an arrow’s axis
Since the axis of an arrow is a vector, we can perform scalar multiplication on it.

• Modify the axis of the red arrow by changing the last line of the program to the following:

```python
arrow(pos=vector(3,2,0), axis=-0.5*vector(3,4,0), color=color.red)
```

Run the program. The axis of the red arrow is now equal to -0.5 times the axis of the white arrow. This means that the red arrow now points in the opposite direction of the white arrow and is half as long. Multiplying an axis vector by a scalar will change the length of the arrow, because it changes the magnitude of the axis vector. The arrow will point in the same direction if the scalar is positive, and in the opposite direction if the scalar is negative.

For the next section, we will only need one arrow. Let’s make the computer ignore one of the “arrow” lines in the program.

• Change the second to last line (the white arrow) to the following:

```python
#arrow(pos=vector(2,-3,0), axis=vector(3,4,0), color=color.white)
```

Note the pound sign at the beginning of the line. The pound sign lets VPython know that anything after it is just a comment, not actual instructions. The line will be skipped when the program is run.

• Run the program. There should now only be one arrow on the screen.

10. Arrows and position vectors
We can use arrows to represent position and relative position vectors. Remember that a relative position vector that starts at a position $A$ and ends at a position $B$ can be found by “final minus initial,” or $B - A$. Do the following exercise:

We want to make an arrow represent the relative position vector of the tennis ball with respect to the baseball. That is, the arrow’s tail should be at the baseball’s (the white sphere) position, and the tip should be at the tennis ball’s (the green sphere) position.

• Write the “pos” of this arrow: `vector(_______, _______, _______)`
11. Naming objects and using object attributes

- Now change the position of the tennis ball (the second sphere in the program)—imagine it now has a z-component, so that the line would now be:

  \[ \texttt{sphere(pos=vector(-3,-1,3.5), radius=0.15, color=color.green)} \]

- Run the program.

Note that the relative position arrow still points in its original direction. We want this arrow to always point toward the tennis ball, no matter what position we give the tennis ball. To do this, we will have to refer to the tennis ball’s position symbolically. But first, since there is more than one sphere and we need to tell them apart, we need to give the objects names.

- Change the “sphere” lines of the program to the following:

  \[
  \texttt{baseball=sphere(pos=vector(2,4,0), radius=0.20, color=color.white)} \\
  \texttt{tennisball=sphere(pos=vector(-3,-1,3.5), radius=0.15, color=color.green)}
  \]

We’ve now given the spheres names. We can use these names later in the program to refer to each sphere individually. Furthermore, we can specifically refer to the attributes of each object by writing, for example, “tennisball.pos” to refer to the tennis ball’s position attribute, or “baseball.color” to refer to the baseball’s color attribute. To see how this works, do the following exercise.

- Start a new line at the end of your program and type:

  \[
  \texttt{print tennisball.pos}
  \]

- Run the program.
- Look at the text output window. Is the printed vector the same as the tennis ball’s position? 

Let’s also give a name to the arrow.

- Edit the last line of the program (the red arrow) to the following:

  \[
  \texttt{bt=arrow(pos=vector(2,4,0), axis=vector(-5,-5,0), color=color.red)}
  \]

Since we can refer to the attributes of objects symbolically, we want to write symbolic expressions for the “axis” and “pos” of the arrow “bt”. The expressions should use general attribute names, like “tennisball.pos”, and not specific vector values, like (-3,-1,0). This way, if the positions of the tennis ball or baseball are changed, the arrow will still point from baseball to tennis ball.
• For arrow “bt”, the final point is the tennis ball’s position. The initial point is the baseball’s position. Write a symbolic VPython expression for the “pos” of “bt”:

\[
\text{pos=}
\]

• Now write a symbolic VPython expression for the “axis” of “bt”. (Remember that a relative position vector that starts at a position \( A \) and ends at a position \( B \) can be found by “final minus initial,” or \( B - A \).)

\[
\text{axis=}
\]

• Now, change the last line of the program so that it uses these expressions for “pos” and “axis”.

• Run the program. Make sure the red arrow still points from the baseball to the tennis ball.

• Change the “pos” of the baseball to (-4, -2, 5). Change the “pos” of the tennis ball to (3, 1, -2). Run the program. Does the arrow still point from the baseball to the tennis ball?

---

At this point, stop and call an instructor to check your work. Do not continue until an instructor has told you it’s OK to go on.

Instructor initials

12. Loops

Another programming concept we will use in the course is a loop. A loop is a set of instructions in a program that are repeated over and over until some condition is met. There are several ways to create a loop, but most often in this course we will use the “while” statement to make loops.

Let’s try using a loop to repeatedly add to a quantity and print out the current value of the quantity.

• Start a new line at the end of your program and type:

\[
t=0
\]

This tells the program to create a variable called “t” and assign it the value of 0.

• On the next line, type:

\[
\text{while} \ t<10:
\]

• Press the “Enter” key. Notice that the cursor is now indented on the next line. (If it’s not indented, check to see if you typed the colon at the end of the previous line. If not, go back and add the colon, then press “Enter” again.)

The “while” statement tells the computer to repeat certain instructions while a certain condition is true. The lines that will be repeated are the ones that are indented after the “while” statement. In this case, the loop will continue as long as the variable “t” is less than 10.

• On the next (indented) line, type:

\[
t=t+0.5
\]
In algebra, "t=t+0.5" would be an incorrect statement, but in VPython, the equals sign means something different than it does in algebra. In VPython, the equals sign is used for assignment, not equality. That is, the line assigns the variable t a new value, which is the current value of t plus 0.5. This means that the first time through the loop, the computer adds the current value of t, which is 0, to 0.5, giving 0.5, and then assigns t this new value of 0.5. The next time through the loop, the computer again adds 0.5 to t, making t equal to 1.0, and so on.

- To show this, on the next line (still indented), type:

```python
print t
```

The last four lines you typed should now look like this:

```python
t=0
while t<10:
    t=t+0.5
    print t
```

- Run the program.

In the text output window, you should see a list of numbers from 0.5 to 10.0 in increments of 0.5. The first number, 0.5, is the value of t after the first time through the loop. Before each execution of the loop, the computer compares the current value of t to 10, and if it is less than 10, it executes the loop again. After the 20th time, the value of t is now 10.0. When the computer goes back to the "while" statement for the next repetition, it finds the statement "t<10" is now false, since 10.0 is not less than itself. Because the condition is false, the computer does not do any more executions of the loop.

To go back to writing statements that are not repeated in a loop, simply unindent by pressing the "Backspace" key.

- Type the following on a new, unindented, line:

```python
print "End of program"
```

Now the last five lines should look like this:

```python
t=0
while t<10:
    t=t+0.5
    print t
print "End of program"
```

- Run the program.

You'll now see the sequence of numbers printed, followed by the text "End of program." The line that prints this text is not in the loop, so the text prints only after the loop is done executing.

At this point, stop and call an instructor to check your work. Do not continue until an instructor has told you it's OK to go on.

Instructor initials__________________
13. Creating a static model (optional)

This section is optional. If there is time remaining, try this section to get more practice with creating and manipulating 3D objects. Follow the instructions below.

First, save your old program if you wish, then start a new one by going to the “File” menu and selecting “New window.” Again, the first line to type in this new window is:

```python
from visual import *
```

The program you will write makes a model of the Sun and various planets. The distances are given in scientific notation. In VPython, to write numbers in scientific notation, use the letter “e”; for example, the number 6.4 x 10^7 is written as 6.4e7 in a VPython program.

Create a model of the Sun and three of the inner planets—Mercury, Venus, and Earth. The distances from the Sun to each of the planets are given by the following:

- Mercury: 5.8 x 10^10 m from the sun
- Venus: 1.1 x 10^11 m from the sun
- Earth: 1.5 x 10^11 m from the sun

The inner planets all orbit the sun in roughly the same plane, so place them in the x-y plane. Place the Sun at the origin, place Mercury at <d, 0, 0>, place Venus at <-d, 0, 0>, and place Earth at <0, d, 0>, where d represents the distance from the Sun to the particular planet.

If you use the real radii of the Sun and the planets in your model, they will be too small for you to see! So use these values:

- Radius of Sun: 7.0 x 10^9 m
- Radius of Mercury: 2.4 x 10^9 m
- Radius of Venus: 6.0 x 10^9 m
- Radius of Earth: 6.4 x 10^9 m

The radius of the Sun in this program is ten times larger than the real radius, while the radii of the planets in this program are 1000 times larger than the real radii.

Finally make two arrows:
1.) Create an arrow representing the relative position of Mercury with respect to the Earth.
2.) Imagine that a space probe is on its way to Venus, and that it is currently halfway between Earth and Venus. Make a relative position vector that points from the Earth to the current position of the probe.

---

**At this point, stop and call an instructor to check your work.**

Instructor initials ___________________
Appendix 10: Modeling motion tutorial, Fall 2003
Modeling motion with VPython

Every program that models the motion of physical objects has two main parts:

1. **Before the loop**: The first part of the program tells the computer to:
   a. Create 3D objects
   b. Give them initial positions and momenta
   c. Create numerical values for constants we might need

2. **The “while” loop**: The second part of the program, the loop, contains the lines that the computer reads to tell it how to increment the position of the objects over and over again, making them move on screen. These lines tell the computer:
   a. How to calculate the net force on the objects
   b. How to calculate the new momentum of each object, using the net force and the momentum principle
   c. How to find the new positions of the objects, using the momenta

We will now introduce how to model motion with a computer. The program you will write will model the motion of the dynamics cart on the track.

1. **Before the loop**

   **Creating the objects**
   - Open IDLE for Python, and type the usual beginning line:

   ```python
   from visual import *
   ```

   First we’ll create a box object to represent the dynamics track that is one meter long.
   - On the next line type the following:

   ```python
   track = box(pos=vector(0, -.075, 0), size=(1.0, 0.05, .10))
   ```

   The `pos` attribute of a box object is the position of the *center* of the box. The `size` attribute gives the box’s length in the x, y and z directions.

   To represent the dynamics cart, we will use a sphere object. (You could use a box if you prefer, but spheres are somewhat easier to deal with.) We will place the cart at the left side of the track to start.
   - On the next line type the following:

   ```python
   cart = sphere(pos=vector(-0.5, 0, 0), radius=0.05, color=color.green)
   ```

   Run the program to see the track and “cart”.

   **Initial conditions**
   Any object that moves needs two vector quantities declared before the loop begins:
   1. initial position; and
   2. initial momentum.

   We’ve already given the cart an initial position of \((-0.5, 0, 0)\) m. Now we need to give it an initial momentum. If you push the cart with your hand, the initial momentum is the momentum of the cart just *after* it leaves your hand. Since the definition of momentum, at speeds much less than the speed of light, is $p = mv$, we need to tell the computer the cart’s mass and the cart’s initial velocity.
• On a new line type the following:

\[ \text{cart.m} = 0.80 \]

The symbol `cart.m` now stands for the value 0.80, the mass of the cart in kilograms.

Now that we have the mass, multiplying the mass by the initial velocity will give the initial momentum. Let's give the cart an initial velocity of \( \langle 0.5, 0, 0 \rangle \) m/s. Then, the initial momentum would be the mass times this initial velocity vector.

• On a new line type the following:

\[ \text{cart.p} = \text{cart.m*vector}(0.5, 0, 0) \]

This now gives the cart an initial momentum value that is \( (0.80 \text{ kg})(0.5, 0, 0) \text{ m/s} = (0.4, 0, 0) \text{ kg m/s} \). The symbol `cart.p` will stand for the momentum of the cart throughout the program.

**Note:** We could have called it the momentum just `p`, or the mass just `m`, instead of `cart.p` or `cart.m`. There are no “built in” physics attributes `p` or `m` for objects like there are built-in geometrical attributes `pos` or `radius`. We can make any attributes we want. It’s useful to create attributes like mass or momentum for objects; this way, we can easily tell apart the masses and momenta of different objects.

**Timestep and time**

To make the cart move we will use the equation \( \Delta \vec{r} = \vec{v} \Delta t \). We need to define a variable `deltat` to stand for the timestep \( \Delta t \). Later in the course you will learn how to choose an appropriate value for `deltat`.

• On a new line type the following:

\[ \text{deltat} = 0.01 \]

• Calculate how far the cart will move in one time step (one execution of the loop) ______________

• How many executions of the loop will it take for the cart to move one meter? ______________

Next we want to set a variable `t`, which stands for time, to start at zero seconds. Each time the program goes through the loop, we will increment the value of `t` by `deltat` (in this case, 0.01 seconds).

• On a new line type the following:

\[ t = 0 \]

2. The “while” loop

**Constant momentum motion**

Now we will create a “while” loop. Each time the program reads through this loop, it will do two things:

1. Calculate the change in the cart’s position and use it to find a new position
2. Increment time `t` by `deltat`
• On a new line type the following:

```python
while t < 3.0:
```

• Press return, and make sure the cursor is now indented on the next line.

This tells the computer to keep executing the loop while time is less than 3 seconds.

In class, you learned that the change in position of an object during a short period of time is given by:

\[ \Delta \vec{r} = \vec{v} \Delta t \]

where \( \vec{r} \) is the object’s position, and \( \vec{v} \) is the object’s velocity. Since the symbol \( \Delta \) means “final minus initial”, we could also write this equation as:

\[ \vec{r}_f - \vec{r}_i = \vec{v} \Delta t \]

where \( \vec{r}_f \) is the final position of the object, and \( \vec{r}_i \) is its initial position. We can then bring \( \vec{r}_i \) to the other side of the equation (add \( \vec{r}_i \) to both sides), giving the equation:

\[ \vec{r}_f = \vec{r}_i + \vec{v} \Delta t \]

Finally, since at low speed \( \vec{p} \approx m \vec{v} \), or \( \vec{v} \approx \vec{p} / m \), we can write

\[ \vec{r}_f = \vec{r}_i + (\vec{p} / m) \Delta t \]

We will use this equation to increment the position of the cart in the program. First, we must translate it so VPython can understand it.

• On the indented line after the “while” statement, type the following:

```python
    cart.pos = cart.pos + (cart.p/cart.m)*deltat
```

Notice how this statement corresponds to the algebraic equation:

```
\[
\vec{r}_f = \vec{r}_i + (\vec{p} / m) \Delta t
\]
```

```
cart.pos = cart.pos + (cart.p/cart.m)*deltat
```

In the program, the final and initial position is written the same way: `cart.pos`. Remember, this is because this really isn’t an equality, but an assignment statement—`cart.pos` is being assigned a new value equal to its old value plus `(cart.p/cart.m)/deltat`.

Finally we need to increment time \( t \).
• On the indented line after the "while" statement, type the following:

\[ t = t + \text{deltat} \]

• Now, run the program

When you run the program, you should see the sphere at its final point. The program is executed so rapidly that the entire motion occurs faster than we can see. We can slow down the animation rate by adding a "rate" statement.

• Place the cursor at the end of the line that reads "while t<3.0:”. Press the “enter” key to insert a new, indented line right before the "cart.pos" line.
• On this new line, type the following:

\[ \text{rate}(100) \]

Every time the computer executes the loop, when it reads "\text{rate}(100)”, it pauses long enough to ensure the loop will take 1/100th of a second. Therefore, the computer will only execute the loop 100 times per second.

• Now, run the program.

You should see the sphere travel to the right at a constant velocity, stopping beyond the edge of the track.

Note: The cart going beyond the edge of the track isn’t a good simulation of what really happens, but it’s what we told the computer to do. There are no “built-in” physical behaviors, like gravitational force, in VPython. Right now, all we’ve done is told the program to make the cart move in a straight line. If we wanted the cart to fall off the edge, we would have to enter lines into the program that tell the computer how to do this.

At this point, stop and have an instructor check your work. Do not continue until an instructor has told you it’s OK to go on.

Instructor initials______________

3. Force and momentum principle

Now we will modify our program to include a force on the cart. This means we will have to add lines to the program that tell the computer that there is a force on the cart, and tell it how to use the momentum principle to change the cart’s momentum due to the force.

Suppose you push a cart, and after leaving your hand it has an initial momentum to the right, as before. But now, the fan on the cart is opposing this initial momentum, so that there is a net force on the cart to the left. You measured this force to be about 0.4 newtons.

• Write the force vector that points to the left (the -x direction) with a magnitude of 0.4 N:

\[ (\text{______, ______, ____}) \text{ N} \]

Let’s use our program to model this situation. First let’s add a line to that creates a force vector.
• In the loop, insert a blank line right after the “rate(100)” statement. Type the following:

\[ \text{Fnet} = \text{vector}(-0.4, 0, 0) \]

**Note:** This force is a constant value—it doesn’t change each time through the loop, unlike the cart’s position. We could have, therefore, written this line before the loop. But in most of the programs we write, force will be changing with time, so force calculations will typically go in the loop.

Next, we need to use the momentum principle. The momentum principle says the change in an object’s momentum in a short time period is equal to the net force on it, times the time interval. In symbols, this is:

\[ \Delta \vec{p} = \vec{p}_f - \vec{p}_i = \vec{F}_{\text{net}} \Delta t \]

where \( \vec{p} \) is the cart’s momentum, and \( \vec{F}_{\text{net}} \) is the net force on the cart. We can rewrite this as:

\[ \vec{p}_f = \vec{p}_i + \vec{F}_{\text{net}} \Delta t \]

We then use this equation in the program, which tells the computer how to calculate a new momentum for the cart.

• Insert a new line in the loop, right after the line that reads “\( \text{Fnet} = \text{vector}(-0.4, 0, 0) \)”.
• On this new line, type the following:

\[ \text{cart.p} = \text{cart.p} + \text{Fnet} \times \text{deltat} \]

Note the similarity between this line and the line that updates the position. Here is how the algebraic equation corresponds to the VPython code:

• Run the program. You should see the sphere move to the right a small distance before changing direction and moving to the left.

• Change the initial momentum of the cart so that it moves a greater distance to the right before changing direction: make the initial speed of the cart 0.9 m/s to the right.

• Run the program. You should now see the sphere move to the right edge of the track before it starts moving to the left.
At this point, stop and have an instructor check your work. Do not continue until an instructor has told you it’s OK to go on.

Instructor initials

4. Motion in two dimensions
Let’s try giving the cart a component of initial momentum in the y-direction (that is, upward). Obviously, the behavior we will see can’t really happen. We just want to demonstrate a situation where the force on an object and its momentum are in different directions.

- Change the initial velocity of the cart to \( \langle 0.9, 0.3, 0 \rangle \) m/s

The sphere now makes a path in the shape of a parabola. Let’s look more carefully at the momentum of the cart during its motion by printing out the momentum vector at each execution of the loop.

- Insert a new line in the loop, just after the line that reads “\( \text{cart.p = cart.p+Fnet*deltat} \)”. Type the following on this new line:

  \[
  \text{print t, cart.p}
  \]

Run the program. The values of time \( t \) and momentum \( \text{cart.p} \) are now printed in the “Python Shell” window at every execution of the loop. Refer to these values when doing the following exercises.

- Using the printed list of momentum vectors, calculate \( \Delta \hat{p} \). That is, pick one of the vectors to be the final momentum, and subtract from it the vector just above it—the initial momentum:

  \[
  \Delta \hat{p} = \hat{p}_f - \hat{p}_i = \langle \ldots, \ldots, \ldots \rangle \text{ kg m/s}
  \]

- Calculate the vector \( \vec{F}_{\text{net}} \Delta t \) (In the program, this would be \( \text{Fnet*deltat} \)).

  \[
  \vec{F}_{\text{net}} \Delta t = \langle \ldots, \ldots, \ldots \rangle \text{ N s}
  \]

- Do your calculations show that the change in momentum \( \Delta \hat{p} \) is equal to \( \vec{F}_{\text{net}} \Delta t \)?

- Look at the printed list of momentum vectors again. Why aren’t the y or z components of the momentum changing?

Be sure you’ve saved copies of your program for future reference.

At this point, stop and have an instructor check your work.

Instructor initials