ABSTRACT

THOMAS, JAMES M. Automated Scaffolding of Task-Based Learning in Non-Linear Game Environments. (Under the direction of R. Michael Young.)

This work described in this dissertation is an attempt to integrate intelligent tutoring capabilities into a framework that can be applied within any exploratory learning game. The goal is to provide dynamic adaptation to the learning needs of an individual student without constraining the autonomy and fun that digital games can offer.

To accomplish this goal, a theoretical framework was designed that employs a novel plan-based knowledge representation to describe both how the game works and what a student can learn in the game environment. This framework was implemented in a system called Annie, that employs a decompositional partial-order planner as its engine. Automated planning has been shown to provide a balance between user autonomy and story coherence within interactive narrative that is similar to the balance between player autonomy and learning progression that is a goal for Annie.

Annie was implemented and evaluated with human learners in a game called FixIt, also created as a component of work for this dissertation. This document describes the theoretical framework, the implementations of Annie and FixIt, the methods and results of the experimental evaluations.
Automated Scaffolding of Task-Based Learning in Non-Linear Game Environments

by
James M. Thomas

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Computer Science
Raleigh, North Carolina
2011

APPROVED BY:

______________________  ________________________
Jon Doyle                  Patrick Fitzgerald

______________________  ________________________
Robert St. Amant          R. Michael Young
Chair of Advisory Committee
DEDICATION

For my Dad.
Jim grew up in Vienna, Virginia, a suburb of Washington, D.C. He played his first computer game in 1976, and later in the same year created his own game: a three-card version of poker written in BASIC. At the University of Virginia in Charlottesville he earned a Bachelors degree in Mathematics, a Masters degree in Computer Science, and met the love of his life, Melissa DeRosier.

Jim wrote mainframe software for IBM’s Myers Corners Lab near Poughkeepsie, NY, before relocating to Cary, NC to work for BNR, Inc. At BNR, Jim helped design and implement the first speech recognition software deployed within a public telecommunication network. Jim was the architect and lead designer of the first Voice over IP system to be used in the public network. He also spent several years as one of just eight lead architects throughout the company on one of the largest (30+ million lines of procedural code) software refactoring projects ever undertaken. Jim spent the next decade as a software design manager, culminating in 1999, when he was named a “top 1%” performer among other awards and bonuses. Jim was anxious for a new challenge and moved into strategic marketing, a function in which he was named a “top 1%” performer in 2000, less than one year after making the transition.

The first sentence of Jim’s one and only application to a Ph.D. program was “I want to get a Ph.D. so I can do more interesting work.” Indeed, Jim has found a mid-life transition to graduate school to be invigorating, interesting, and challenging. Jim was supported by a fellowship from the National Science Foundation’s Graduate Research Foundation in Computer Science, with a specialization in Artificial Intelligence. Jim’s enthusiasm for research and A.I. led him on an circuitous route through planning language metatheory, interactive narrative,
digital games, and ITS before settling on a thesis topic that combines elements of each of these areas of study.
ACKNOWLEDGEMENTS

Foremost, I would like to thank my committee members, Jon Doyle, Robert St. Amant, and Pat Fitzgerald, and my advisor and committee chairman, R. Michael Young for their guidance, attention, and patience. Michael Young has helped me develop as a researcher and as a scientist down a path I could not have found on my own. Also, I greatly appreciate working with James Lester, who helped me connect to the ITS community in innumerable ways.

I also appreciate the support and camaraderie of all my fellow graduate students in the Liquid Narrative and Intellimedia research groups, who never failed to accept me despite my youthful enthusiasm and occasional immaturity. Special thanks are due to Mark Riedl, Arnav Jhala, Brad Mott, Jon Rowe, Seung Lee, Alok Baikadi, Julius Goth, Jen Robinson, Justin Harris, Tommy Vernieri, Pat Cash, Joe Winegarden, Yuna Cheong, Leo Bae, James Niehaus, Sam Munilla, Mike Dominguez, Stephen Ware, Matt Fendt, Julio Bahamon, Brad Cassell, Charles Bevan, Stephen Roller, and the rest of the folks who have made the years go by too fast in the DGRC lab.

I would like to thank my parents for encouraging me to put myself in challenging situations. I will never forget when I told my Dad that maybe I shouldn’t leave my current position because I gained 12 years of work experience in that area he asked, “Is it really twelve years of experience or just one year repeated twelve times?”

Immeasurable thanks go to my family, my wife Melissa and sons Jefferson, Benjamin, and Lincoln, and my Mom, who is always on my side. I would not have made it without your support. And thanks to my father, who returned to school in his fifties to earn a Master’s Degree with a 4.0 GPA, for the continuing inspiration. I know you would have loved this, Dad.
# Table of Contents

List of Tables ................................................. ix

List of Figures ................................................. x

Chapter 1  Introduction ......................................... 1
  1.1 Problem Statement ....................................... 1
  1.2 Background ........................................... 2
  1.3 A Note On Terminology ................................... 4
  1.4 Automated Scaffolding .................................. 6
  1.5 Interactive Narrative ................................... 8
  1.6 Research Aspirations and Contributions ............... 10
    1.6.1 Aspirations ....................................... 10
    1.6.2 Contributions .................................... 11
  1.7 Readers’ Guide ......................................... 12

Chapter 2  Related Work ....................................... 14
  2.1 Automated Scaffolding .................................. 14
    2.1.1 Scaffolding in Exploratory ITSs .................. 16
    2.1.2 Scaffolding in “Mainstream” (Non-Exploratory) ITSs 17
    2.1.3 Scaffolding and Alternative Techniques in Digital Games 19
  2.2 Interactive Narrative and Learning .................... 25
    2.2.1 Steve: Soar Training Expert for Virtual Environments 25
    2.2.2 Mission Rehearsal Exercise ....................... 26
    2.2.3 U-DIRECTOR ..................................... 27
  2.3 Plan-Based Representations and Task-Based Learning .... 28
    2.3.1 Discourse-focused Tutorial ....................... 29
    2.3.2 Agent Learning Approaches ....................... 29

Chapter 3  Design Overview ................................... 31
  3.1 System Architecture .................................... 33
  3.2 Operational Overview ................................... 36
    3.2.1 System Initialization .............................. 36
    3.2.2 Execution Cycle: Overview ....................... 37

Chapter 4  Detailed Design ................................... 42
  4.1 Annie’s Inputs ......................................... 43
  4.2 System Initialization ................................... 48
  4.3 Execution Cycle ....................................... 50
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Belief Strength Levels</td>
<td>46</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Student Inaction</td>
<td>79</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Action Failure - Unthreatened Precondition</td>
<td>80</td>
</tr>
<tr>
<td>Table 4.4</td>
<td>Action Failure - Unresolved Threat</td>
<td>81</td>
</tr>
<tr>
<td>Table 4.5</td>
<td>Success - Fatal Flaw</td>
<td>82</td>
</tr>
<tr>
<td>Table 4.6</td>
<td>Irrelevant Action</td>
<td>83</td>
</tr>
<tr>
<td>Table 4.7</td>
<td>Action Success - Suboptimal Plan</td>
<td>85</td>
</tr>
<tr>
<td>Table 4.8</td>
<td>Action Success - Correct Beliefs</td>
<td>86</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Per-Student Comparative Diagnostic Accuracy</td>
<td>98</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Treatment Group Assignments</td>
<td>107</td>
</tr>
<tr>
<td>Table 5.3</td>
<td>Student Task Completion Percentages</td>
<td>109</td>
</tr>
<tr>
<td>Table 5.4</td>
<td>Student Challenge Completion Times</td>
<td>111</td>
</tr>
<tr>
<td>Table 5.5</td>
<td>Hypothesis 2A: Challenge Completion Time - Correlation</td>
<td>112</td>
</tr>
<tr>
<td>Table 5.6</td>
<td>Correlations: Treatment Level vs. Completion Time</td>
<td>113</td>
</tr>
<tr>
<td>Table A.1</td>
<td>Processes: Fixed Attributes</td>
<td>142</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1.1 Scaffolding and the Zone of Proximal Development . . . . . . . . . . 7
Figure 1.2 Task Decompositions for Scaffolding . . . . . . . . . . . . . . . . . . 9

Figure 3.1 System Architecture . . . . . . . . . . . . . . . . . . . . . . . . . . 34
Figure 3.2 Execution Data Flows . . . . . . . . . . . . . . . . . . . . . . . . . . 39

Figure 4.1 Remediation Consideration . . . . . . . . . . . . . . . . . . . . . . . 54
Figure 4.2 Student Inaction . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 79
Figure 4.3 Unthreatened Precondition . . . . . . . . . . . . . . . . . . . . . . . 80
Figure 4.4 Unresolved Threat . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 81
Figure 4.5 Fatal Flaw . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 83
Figure 4.6 Irrelevant Action . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 84
Figure 4.7 Suboptimal Plan . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85
Figure 4.8 Diagnosing Correct Beliefs . . . . . . . . . . . . . . . . . . . . . . . 86

Figure 5.1 FixIt: Top Down View . . . . . . . . . . . . . . . . . . . . . . . . . . 89
Figure 5.2 FixIt: Mouse Droid (Center) Emerging From Process Panel . . . . . 91
Figure 5.3 Pilot Evaluation Instructions (page 1) . . . . . . . . . . . . . . . . . . 94
Figure 5.4 Pilot Evaluation Instructions (page 2) . . . . . . . . . . . . . . . . . . 95
Figure 5.5 Pilot Evaluation Self-Assessment (page 1) . . . . . . . . . . . . . . . . 116

Figure A.1 FixIt: Level Views . . . . . . . . . . . . . . . . . . . . . . . . . . . . 137
Figure A.2 FixIt: Important Actors . . . . . . . . . . . . . . . . . . . . . . . . . 141
Figure A.3 FixIt: System Status Indicators . . . . . . . . . . . . . . . . . . . . . 145

Figure B.1 Operator Library XML Excerpt . . . . . . . . . . . . . . . . . . . . . 154
Figure B.2 Plan XML Excerpt . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155
Figure B.3 Portion of Plan Space Graph from Pilot Evaluation . . . . . . . . . . 157
Figure B.4 A Plan Graph from Pilot Evaluation . . . . . . . . . . . . . . . . . . . 158
Figure B.5 Plan Graph Detail . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 159

Figure C.1 Main Evaluation PreAssessment (page 1) . . . . . . . . . . . . . . . . 163
Figure C.2 Main Evaluation PostAssessment (page 1) . . . . . . . . . . . . . . . . 164
Figure C.3 Main Evaluation PostAssessment (page 2) . . . . . . . . . . . . . . . . 165
Chapter 1

Introduction

1.1 Problem Statement

Intelligent Tutoring Systems (ITS) have demonstrated impressive teaching results in a variety of knowledge domains [19, 86, 92], largely by leveraging detailed cognitive models that allow for fine-tuned adaptation to the needs of particular learners. However, because many traditional ITSs “follow the principle that the learner should be presented with a restricted set of activities to work on under the supervision of a computerized coach [10, p. 269]”, it has proven difficult for the field to converge on general approaches to guide learning within exploratory environments. For the purposes of this paper, an exploratory environment is defined as one that allows the student to choose from a wide variety of activities throughout the learning experience. Such exploration is a core component of ITS systems variously described as supporting “scientific discovery learning” [96, 64, 22], “inquiry learning” [23, 84, 31], “open” [16, 10, 50], or “exploratory” [56, 11] learning. A key challenge in these type of ITSs is to balance guidance with student exploration and “in such a way that learning is supported effectively, but the inquiry process is not reduced to following cookbook instructions [84, p. 112].”
In contrast, digital games have developed innovative techniques for guiding people through exploratory environments. Despite increasing complexity, games are increasingly easy to learn, because games must be easy to learn to sell well enough to recoup their massive production investments. Researchers have catalogued dozens of learning principles in digital games [28, 60], but they are most frequently implemented as fixed alternative paths tightly woven into the structure of particular games. These multiple options afford a non-linear game structure that rarely coincides with deep modeling of student knowledge or dynamic adaptation to individual users.

This dissertation describes a general conceptual framework for guiding task-based learning in an exploratory environment. The aim is to combine the exploratory learning mechanisms of digital games with the deep student models of intelligent tutoring systems. In addition, the dissertation describes an implemented system built on that framework and a stand-alone game-based learning environment, both developed by the author to evaluate the effectiveness of the introduced approach. Two formal evaluations of the effectiveness of the system in working with learners in the game-based learning environment are then presented. Although some positive effects were found on learner performance, several areas for future improvements are identified.

1.2 Background

The motivation for creating and evaluating this work was drawn from intriguingly similar problems noted across several fields of inquiry. First, modern digital games have developed some extremely powerful teaching techniques unique to that medium. Researchers have created successful game-based learning environments [10, 49], but critics have charged that a facile
attempt to combine learning with games often either “sucks the fun out” or “sucks out the learning” [61]. Postman goes so far as to argue that learning and entertainment are mutually exclusive: “no one has ever said or implied that significant learning is effectively, durably and truthfully achieved when education is entertainment [59, p. 146].” Nevertheless, a broad community of serious games researchers [12, 97, 83, 47, 70, 36] continues to work to harness the rapidly evolving technologies of digital games to empower teaching in many different domains.

Meanwhile, Intelligent Tutoring Systems (ITS) have demonstrated impressive teaching results in a variety of knowledge domains [19, 87, 2, 24, 29, 48, 3, 63], largely by leveraging detailed cognitive models that allow for precise adaptation to the needs of particular learners. Exploratory learning has long been an active interest of ITS research, especially in the realms of scientific inquiry [24, 96, 64, 22, 31] and mathematics [10, 17]. But as Quintana et. al. point out, despite “individual successes, accumulation of both theory and craft knowledge about scaffolding design has been difficult. The field has not converged on a common framework [...] making it difficult to synthesize claims and results across contexts [62, p. 339].” Although the authors present a synthesis of approaches to scaffolding within tutoring systems that aid scientific discovery, it is more an analysis of what has come before than a road map for generating future systems.

A third field in which a similar problem is found is interactive narrative, where a balance must be struck between overall narrative coherence and player autonomy [94, 5]. An interactive narrative should allow for exploration that can change plot points, just as an exploratory learning system should allow the student to attempt learning tasks in varying order. Likewise, both must balance this need for user autonomy with higher level goals: story coherence in the case of interactive narrative, and learning objectives in the case of exploratory ITS. This
dissertation builds on work in interactive narrative that addresses this balance.

1.3 A Note On Terminology

The title of this dissertation, “Automated Scaffolding of Task-Based Learning in Non-Linear Game Environments,” employs several carefully chosen terms whose usage may not be familiar to all readers. “Scaffolding,” in the context of the sub-field of Artificial Intelligence known as Intelligent Tutoring Systems (ITS), is the process of presenting help, often in the form of text-based prompts, to users of computer-based learning systems. Although the terms “scaffold,” and “scaffolding” are found extensively throughout the ITS literature, they are not consistently defined. For the purposes of this dissertation, a scaffold is defined as an atomic unit of help provided to the learner. The process of scaffolding is the optimal selection of the most appropriate scaffold to use with the student at any given point in time.

The intended denotation of the phrase “Non-Linear Game Environments” is the subset of digital or computer games that do not prescribe a linear path the player must take through the game. Instead, such games allow players autonomy in exploring or changing the state of objects in the game environment. Most modern three-dimensional games fall into this category, but also included are non-3D games that promote player autonomy, such as text-based adventure games [38]. As will be shown, automatically generating scaffolding within non-linear game environments presents new and difficult challenges.

Finally, the phrase “task-based learning” is meant to constrain the scope of learning addressed by the this work to the actions within a particular domain. Many domains within the natural sciences, such as RNA transcription, celestial mechanics, photosynthesis, thermodynamics and the Krebs cycle feature the complex hierarchies of inter-related actions that are
the focus of this work. Not only are these domains fairly difficult to learn through traditional sequentially organized instructional media (e.g., books), but they are particularly well-suited to representation in a exploratory learning environments.

The conceptual framework described in this work is designed to enable systems to be implemented that automatically generate scaffolding within learning environments that grant a high degree of task-selection autonomy, encouraging knowledge acquisition through student-initiated exploration and discovery. The framework is intended to be generally applicable across a wide range of domains because its core abstractions are the actions the student or the system can use to change the state of the environment. It builds a student model based on the individual student’s sequence of action attempts. The student model guides the generation of scaffolds that also take the form of actions performed in the environment. It is left to the domain author to map these actions to domain-relevant concepts, and to specify the learning objectives in terms of the student’s performance and understanding of the various actions available in the world.

The system implementing this framework is named “Annie,” in honor of Anne Sullivan who taught Helen Keller the rules of verbal language when Helen could not hear, speak, or see. The key evaluation criterion for this system is its effectiveness in achieving learning objectives. The key question addressed by this work is whether or not a system whose student model built entirely in terms of the actions available in the world can be sufficiently expressive to provide useful guidance. If Annie can enhance learning effectiveness, it may highlight a new path toward future systems that can combine the exploratory learning mechanisms of digital games with the fine-grained adaptation of intelligent tutoring systems. The next few sections provide a brief overview of the two areas of research on which Annie most heavily relies: automated
1.4 Automated Scaffolding

"Scaffolding" is a term first coined as a metaphor for learning support processes by Wood, Bruner, and Ross [90]. In studying human tutors working with 3-5 year olds, the authors found that three year olds received the most support from the human tutor, and five year olds the least. The word "scaffold" was chosen to convey that the supports provided by the tutor were temporary constructs to be removed or "faded" as the student gained proficiency, just as scaffolds used on construction projects are temporary supports that are built up and then removed over the course of work.

“Scaffolding” quickly became a dominant theme in educational research partly through its links to Lev Vygotsky’s theories of learning [91, 75, 8]. Vygotsky defined the place where learning is most efficient as the zone of proximal development (ZPD): “the distance between the actual development level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or collaboration of more capable peers [89, p. 86].” New competencies can only be generated in the ZPD through collaboration in actual, concrete, situated activities with a more capable peer. With enough assisted practice, the child internalizes the strategies and language for completing this task, which then becomes part of the personal problem-solving repertoire.

Within the Intelligent Tutoring System community, Murray and Arroyo [53] define scaffolding as exactly the assistance required to keep a student in the ZPD, writing: “We want to give assistance in order to keep the learner at their leading edge - challenging, but not overwhelming them [53, p. 751].” For these authors, “the ZPD is neither a property of the learning environ-
ment nor of the student; it is a property of the interaction between the two [53, p. 751].” The role of the tutor is to ensure that the learner is not overly confused by challenges that are too difficult or overly bored by challenges that are too easy. Either condition can lead to frustration, distraction or disengagement. As skills are mastered, scaffolding assistance is “faded” to ensure that learners do not become bored.

Digital games researchers tout a ZPD-like construct they call the “optimal gameplay corridor.” Industry pioneer Hal Barwood [6] describes this corridor with a chart strikingly similar to those used by Masuch [42] and ITS researchers. Barwood credits psychologist Mihaly Cskszentmihaly’s best-selling “Flow“ [21] books for the genesis of the construct within the game industry. In both venues, an upward-sloping region represents the ideal learning path, where anything steeper can lead to confusion and anything more shallow can lead to boredom.

Figure 1.1: Scaffolding and the Zone of Proximal Development
Figure 1.1 depicts the zone of confusion in progressively deeper shades of red toward the upper left corner of the chart and the zone of boredom in progressively deeper shades of blue toward the bottom right corner of the chart. The diamonds represent interventions that either add scaffolding (the “S+” labels) to prevent the learner’s path from straying too far toward confusion, or fade/remove scaffolds (“S-”) to stave off boredom.

Although digital games and Interactive Tutoring Systems seem to have both embraced the concept of scaffolding, their implementation strategies are divergent. Digital games tend to favor user initiative and autonomy at the expense of fine-grained adaptation. ITS systems adapt to subtle details of learning concepts, but tend to constrain student initiative and autonomy beyond what is typical of digital games. Dynamically balancing between user control and a coherent experience is a problem common to the related field of interactive narrative.

1.5 Interactive Narrative

The field of interactive narrative studies the automatic generation of stories within virtual worlds in which human users interact with one or more computer controlled agents [40, 14, 43, 76]. The advantage of interactive narrative is that it can adapt to different choices the user may make in the environment. A persistent challenge in these systems is the narrative paradox: how to reconcile the needs of the user who is now potentially a participant rather than a spectator with the idea of narrative coherence. [4]. More specifically,‘two key challenges posed by narrative-centered environments for exploratory learning are (1) supporting the hypothesis-generation-testing cycles that form the basis for exploratory learning, and (2) orchestrating all of the events in the unfolding story to support appropriate levels of student motivation, engagement, and self-efficacy for effective learning [49].
The Annie system described in this dissertation builds on one approach of balancing narrative and user goals first described in the Mimesis system [69]. Mimesis generates plans for actions of agents in a story world based on hierarchical task decompositions and discrete causal requirements. Mimesis simultaneously solves for plot coherence and character believability, mediating the actions of the human user through a combination of accommodations (when re-planning can preserve narrative goals) and interventions (where unanticipated and harmful interactions posed by user activity would be fatal to narrative goals).

Annie extends and repurposes the hierarchical mediation of Mimesis into a learning context. For example, figure 1.2 shows a simplified example of how a hierarchical plan can be exploited to automatically generate scaffolding. The dash-bordered rectangles represent alternative de-

Figure 1.2: Task Decompositions for Scaffolding
compositions of a single tutorial task. These sub-tasks may consist of any combination of student actions (ovals), or tutorial actions (diamonds). Based on the current state of the student model, Annie selects one of these decompositions and initiates the first action (T1, T2, or T4). The easiest decomposition, shown at the bottom of the figure, is the one where each student action is prompted by a unique system action likely providing some “bite-sized” advice to the student. More complex for the student are decompositions like that shown at the top of the figure, where a single system action is expected to kick off a series of appropriate student actions.

1.6 Research Aspirations and Contributions

This research goals described in this dissertation can be categorized in two ways. One set of goals represent actual contributions that are implemented and evaluated as described in this document. Another set of goals, better labelled as aspirations, are described in the conceptual framework and influential in the design and implementation of Annie, but not yet empirically evaluated.

1.6.1 Aspirations

The following are research goals for this work are best described as aspirations.

An implementation architecture independent of any particular domain or game One of the motivations behind the design of this conceptual framework is that it should be independent of any particular game or domain. The goal for this separation is to promote reuse and comparison between different methods of maintaining a student model and using the student model to automatically generate scaffolding and fading. Given that Annie has
been evaluated in only a single game and a single domain, neither form of independence has yet been demonstrated. Therefore, it is more appropriate to describe this research goal as an aspiration than as a contribution to the research literature. A high priority item on the list of future work is to evaluate Annie in a new game and domain to demonstrate this independence.

**External libraries of diagnostic and remediation templates** The libraries of diagnostic and remediation templates described in Chapter 4 are intended to be separable from the core algorithms in Annie, although in the current implementation they are not. With better understanding of the needs of serious game designers who would use such templates, future work will endeavor to encapsulate these templates in externally manipulable files.

### 1.6.2 Contributions

This dissertation makes a number of research contributions:

**A conceptual framework.** Specifically, a conceptual framework to unify the representation of student knowledge and tutorial reasoning in task-based learning environments. The key decision enabling this work to simultaneously address the needs of the learner and the goals of instruction is to use a common framework to describe and reason about scaffolding alternatives and the needs of the learner.

**A working system that implements the framework.** This dissertation describes a functioning system, called Annie, which implements the conceptual framework. A finely detailed description of the design decisions within Annie, provided in Chapter 4, gives insight to both the methods chosen for implementation and some of the rejected alternatives.

**Plan-based reasoning algorithms for diagnosis of task-based misconceptions.** Annie can rea-
Annie is able to generate metrics to rank the urgency between proximate learning objectives.

An learning environment. A completely independent game and underlying learning environment were constructed to support evaluations of Annie’s effectiveness. Lessons learned in the creation of this game, named “FixIt”, have informed the design of Annie, methods for evaluating Annie, and plans for future learning environments.

Two formal evaluations with human learners. Two formal evaluations of Annie within the FixIt game were held several months apart. The first evaluation compared the accuracy of the automatically generated student model within Annie to that derived by an expert human observer of the performance of subjects within the game. The second evaluation studied the effectiveness with which Annie was able to guide students using the FixIt game through automatically generated scaffolding.

1.7 Readers’ Guide

The remainder of the document is organized as follows. Chapter 2 summarizes related research in the fields of Intelligent Tutoring Systems, digital games, interactive narrative, and plan-based knowledge representations. In Chapter 3, an overview of the design of the conceptual framework supporting the Annie system is described and Chapter 4 describes the implementation details of Annie, including the plan reasoning algorithms. Chapter 5 presents the methods and results of the two experimental evaluations of Annie. Chapter 6 discusses current limitations and future work, and Chapter 7 offers a conclusion. In addition, an extensive set of Appendices,
liberally referenced throughout the text, help to keep a catalogue of supporting information co-located for easy reference without breaking the flow of ideas in the main document.
Chapter 2

Related Work

The work described in this dissertation builds on several overlapping strands of research. Although these have been grouped into three categories for the purposes of this commentary, these categorizations are by no means canonical.

The first category includes descriptions of various approaches to automated scaffolding from mainstream ITS research to the more specialized approaches in exploratory ITSs to the types of scaffolding found in digital games. The second category of related work is the field of interactive narrative. A special focus is given to recent forays into narrative-centered learning. Task-based learning and plan-based knowledge representations which are at the core of this work are grouped into the third and final category.

2.1 Automated Scaffolding

As described earlier, scaffolding is not defined precisely or consistently within intelligent tutoring. Pea laments “I am perhaps not the only one who feels that the concept of scaffolding has become so broad in its meanings in the field of educational research and the learning sciences
that it has become unclear in its significance [...] Perhaps scaffolding has become a proxy for any cultural practices associated with advancing performance, knowledge, and skills whether social, material, or reproducible patterns of interactivity (as in software systems) are involved [57].”

Similarly, a 2005 special issue of *Instructional Science* entitled “Scaffolding Self-regulated Learning and Metacognition: Implications for the Design of Computer-based Scaffolds,” found a wide distribution of the ways in which “scaffolding” was expressed in just the seven papers included in that issue. Some scaffold only when learners explicitly request help, others only when a student makes a mistake. Scaffolding may take the form of dynamic adaptation to individual needs or it may be a static calibration that sets a difficulty level for some duration of the learning exercise. Often systems only add but never remove scaffolds, a process referred to as fading. Even the nature of what is being scaffolded varies from domain knowledge, to learning about the system, to learning about one’s own learning.

The looseness of the definitions of scaffolding within the broader learning sciences community complicate a precise definition of what constitute “automated scaffolding.” One could argue that almost any form of user assistance supplied by a automated system is “automated scaffolding.” A sufficiently loose definition of scaffolding could characterize entire field of HCI as providing automated user support. In practice, however, these extreme definitions have not held sway. As will be shown, the traditional or “mainstream” ITSs have build on what can be argued is an overly narrow view of scaffolding through discourse. Exploratory ITSs have employed a broader definition, but have little consensus about what forms of scaffolding are most useful. What passes for scaffolding in digital games [28, 60] is often hard coded into the game and rarely adaptive to particular users. Nevertheless, each of the varied approaches to
automated scaffolding can bring something to bear on the work in this dissertation.

2.1.1 Scaffolding in Exploratory ITSs

In their exhaustive and extensively cited survey [24], de Jong and van Joolingen compared the types of learner supports offered in exploratory learning environments, specifically, environments geared toward scientific discovery learning. The authors first noted that a number of similar difficulties encountered by learners are common across different systems. For example, they report several cases in the literature where students fail to look for evidence that might not support a hypothesis, incorrectly retain hypotheses in spite of negative results, misinterpret data, and poorly plan and manage their experiments.

The authors proceed to identify eight categories of scaffolding that address these common problems, citing examples from the same literature. For example, the “Support for Design of Experiments” category identifies a system that would offer hints such as “vary only one factor at a time.” Although each of the identified categories contains a few examples of similar types of scaffolding, the overall picture is of a field with little consensus about which types of supports are useful or most important.

Although Quintana et. al., [62] purport to describe a scaffolding “framework” for exploratory ITSs, it is not much more prescriptive to a system builder than a catalogue of design choices excerpted from existing systems. Because there is so little consensus about scaffolding in the field of exploratory ITSs, both this article and the deJong and vanJoolingen survey, merely attempt to categorize and synthesize the differing approaches of existing projects. Quintana et. al. do go a step further in that the framework is organized around a set of principles from learning theory, beginning with a division into the three categories of “Sense Making,”
“Process Management” and “Articulation and Reflection.” Nearly twenty distinct scaffolding strategies are described across these three categories. For example, scaffolding strategy “3A” is to provide students with multiple views of experimental data, while strategy “7C” aids students in generating self-explanations. Although this can be seen as slightly more prescriptive than the earlier survey, choosing from a menu of high level strategies does not address the key issues to be resolved in building a system.

Perhaps the strong thread of verbal discourse which underlies most ITS research has focused the definition of scaffolding more narrowly than is seen in exploratory systems which have fewer shared communicative constructs on which to ground a common approach. In fact, the ITS field has yet to settle upon a common approach to integrating instructional scaffolding with exploratory learning, “in such a way that learning is supported effectively, but the inquiry process is not reduced to following cookbook instructions [84].” Providing this common general approach is exactly one of the goals of Annie.

2.1.2 Scaffolding in “Mainstream” (Non-Exploratory) ITSs

One could argue that most ITS research is primarily concerned with scaffolding in one form or another. Therefore, this is not the place to chronicle the evolution of scaffolding within all of ITS. What is important is to draw a distinction between the fairly narrow consensus about scaffolding seen in exploratory ITS to the fairly broad consensus of the wider field. Across different types of domains, a common set of user assistance methodologies seems to recur. Much of the work in ITS is focused less on the particulars of the assistance, and more on how much assistance should be provided in which situations within the tutorial progression.
For example, Murray and VanLehn compared a “fixed strategy” tutor to one that selects ITS hint strategies on a decision-theoretic basis. The types of assistance were the same for both tutors, the only difference was which particular types of assistance were selected at particular times. There were four different kinds of assistance: prompt, hint, teach or do. An excerpt of the description of these strategies is included below, because these do characterize a fairly representative sample of scaffolding techniques in traditional ITSs.

The **prompt** message pointed out pertinent information that was already available in the interface without providing any new information. The **hint** message provided partial information about the step - not enough to teach the student how to do the step but perhaps enough to either remind the student how to do the step or help the student figure it out. The **teach** message provided all the information that the student needed to understand the domain rule related to the step, including at least one example, and thus to help the student complete the step correctly by learning the rule. The **do** message told the student exactly what to do for the current step (e.g., what text to enter) without teaching anything about the related rule. These help messages are ranked in order of increasing explicitness about what to do for the current step, from prompt (the least explicit,), through hint, teach, and do (the most explicit) [52].

Consistent with prototypical ITS strategy, help is provided whenever the student makes two errors or explicitly requests help. The fixed strategy tutor “followed a strong successive explicitness constraint: It always provided a provided a help message that was minimally more explicit than any help already provided. In other words, first it provided a prompt message, then hint, then teach, before bottoming out with do [52].”
AutoTutor [29] employs a sophisticated model of the typical dialogue elements between tutors and a learners to among the most nuanced forms of verbal assistance found in ITS. Feedback can be positive “That’s right,” negative “Not quite,” or neutral “Uh-huh.” Hints may be “pumps” for more information “What else?,” or for specific information. Hints may take the form of questions about content, assertions about content, or specific corrections to erroneous content supplied by the student. AutoTutor leverages the discourse medium to provide not only information, but emotional support, motivation, and feedback.

Not all of scaffolding in ITS is associated with types of hints. Murray and Arroyo [53] describe scaffolding as something applicable not only to the provision of hints, but also to the sequencing of content, and the provision of feedback and opportunities for practice. They further describe “hints” as “problem solving assistance that gives information of focuses attention in ways that improve the chances that the learner will be able to solve a problem.” The study in their paper looked at providing hints at two different levels of granularity “concrete” or “formal.” These in turn correspond to Piaget’s [58] stages of development, where more abstract, formal reasoning is a later development for most learners. They found that more advanced learners were more efficient when the system first supplies formal hints, and less advanced learners are more efficient with concrete hints.

2.1.3 Scaffolding and Alternative Techniques in Digital Games

Digital games have been recognized for applying innovative and effective learning principles [28, 60] to guide the player’s experience. The current generation of gamers do not read game instruction manuals, but simply start playing a new game with the expectation that the game itself will teach the nature of the domain, the intricacies of the user interface, and the learning and/or performance goals the game intends the user to achieve. Games that do not meet or ex-
ceed this expectation do not sell, so market pressure is a powerful force behind the continuous improvement of learning in games. Because digital games often feature exactly the types of exploratory interfaces educators wish to exploit to inspire deep learning, it is prudent to search for examples of effective scaffolding techniques within digital games.

Scaffolding in digital games is less dynamic, less adaptive, and less fine-grained than scaffolding in ITSs. Where ITS research strives to adapt systems to the needs of individuals, the game industry designs a one-size-fits-all system for an entire population (a.k.a. “target market”). This requires vast expenditures of time and money, expenditures that keep increasing as the industry evolves. For example, during the design of Halo 2, over 400 test gamers were brought into Bungee studios for more than 2000 hours of meticulously recorded and analyzed game sessions. Analyses included “heat maps” depicting how much time the different players spent in different parts of each level, and where players failed, to discover “choke points” in the design of the game. In the less than three years that transpired between the release of Halo 2 and its sequel Halo 3, the expense of this testing rose by 50%, requiring over 3000 hours of analysis of 600 gamers [81].

This play testing was crucial to learning “if and when players are getting bored or (as is more often the case) frustrated,” because “the goal every developer aims for, is an experience that keeps players in a flow state - constantly surfing the edges of their abilities without bogging down [81].” An additional aspiration of game design is to encourage players to move along an intentionally circuitous route to incorporate experiences that positively affect the player’s enjoyment. Game designers refer to the circuitous route as the golden path and the most direct route spine. “The spine of any game consists of events that are absolutely mandatory [...] those elements of the story the player is guaranteed to experience. Most games have
an entirely linear spine, or a spine that supports a small amount of flexibility in terms of the sequence of events but no variation in which events are involved in completing the game and its story [7].” In contrast, the golden path contains additional, non-mandatory game elements that enhance other aspects of the player’s experience. Perhaps equally important, however, the golden path enhances the player’s sense of agency over the events that occur during gameplay, and helps disguise the essentially static structure of the underlying spine. Designers aim to coerce players to take less-direct but more fulfilling routes through the game. Gee describes a particularly relevant example of this coercion in his description of a training episode of the game *Tomb Raider*. The designers of this game coerce the player off the the game spine being described by the in-game tutor character “Von Croy” onto a golden path that enhances the player’s identification with their avatar (Lara), while enhancing the overall sense of fun.

At one point Von Croy ordered Lara to jump across a cavern; in doing so, she fell in the water below, due to my ineptness in controlling her (via the computer’s keys). She can climb back up again and try the jump again (indeed, she needs to do this to follow Von Croy and eventually complete the episode). But, low and behold, as I (Lara) swam toward land, I (Lara) discovered a golden skull in the water. A player cannot help but think: What if I purposely disobey orders and jump and climb other than where I am told? What other good things will I find? Soon one is just a bit more like the willful and spoiled Lara herself (and practicing yet more jumps and climbs). In such video games, players get practice in trying out new identities that challenge some of their assumptions about themselves and the world. A good science class should do the same [28].

This example is instructive on several points. First, it shows that game designers have more in mind than simply pushing players through the game as quickly as possible. To achieve
other goals (e.g., “fun,” or avatar identification), designers actually steer players off the most expedient path. [72] In an ITS context, other goals might include affect management, or even pedagogical materials that are not directly relevant to the task at hand. Second, this form of guidance is suspiciously brittle. If the player succeeds in this initial jump, the golden skull remains undiscovered. If the game structure could be made to adapt to individual performance differences, as is common in ITS, perhaps another method for profitably disobeying Von Croy could be generated dynamically and the skull could be moved to that path. Third, because of this brittleness, designers must spend a lot of time to calibrate the fixed structure of the game to the target population so that content like the golden skull is not wasted on players who never find it.

The static structure of digital games has deep historic roots. In the early 1980s, when digital games had already evolved into the many different genres evident today, industry pioneer Chris Crawford provided a unifying view of game progression, flatly stating: “Games are not sequential, they are branching tree structures [20].” By “tree structure” Crawford refers to a directed graph where each node represents a game state, arcs are actions that result in state transitions, and nodes with multiple outgoing arcs correspond to choices. Although the scale and visual sophistication of today’s games dwarfs those of the early era, it is still the case that the underlying event structure is fairly simple. The problem with this underlying structure is that it is statically defined at design time. This requires a time-consuming and expensive play-testing effort to calibrate the game to a population (target market) rather than to individuals. As the games industry has grown to eclipse the revenues of the film industry, it has acquired Hollywood’s devotion to proven formulae and disdain for risk. This makes it even more unlikely that the static branching so integral to the conventional practice of game design will evolve any faster than it has over the past several decades. Thus, it is highly unlikely that game design
industry will develop accommodations for extensive user modeling and dynamic adaptation in the near term.

Nevertheless, an impressive variety of scaffolding techniques are found in nearly all games, including a learning principle Gee has identified as “staggered instruction” [28]. As games have become more complex, designers can no longer afford to tell the player everything they need to know at the beginning of the game. It would take too long and it would be difficult for the player to remember it all. Instead, a bootstrapping approach is followed. The player is given just enough overt information and training to get started. As new challenges are encountered, the player is given additional information just in time to confront those challenges. “In essence a game manual has been spread throughout the early episodes of the game, giving information when it can be best understood and practices through situated experience.” Games often employ ITS-like hints on demand, and newer games sometimes will present spontaneous hints to the player. However, in contrast to ITS systems, these hints are often not closely linked with what the player is doing at that instant. Also, these hints are usually statically defined “canned” messages that are identical for every user regardless of play history.

A key element of game architecture that enables staggered instruction is that games are designed as a hierarchy of circumscribed domains and sub-domains. In exploratory games, the sub-domains often conform to geographic boundaries. For example, the player begins the game in a “safe” area of the landscape devoid of any hostile entities and cut off from the rest of the game. Usually, the door or bridge that separates this “training ground” from the rest of the game will remain closed until the player has demonstrated proficiency in the basic survival skills to move on to the next, more challenging area. But modern games are often clever enough to disguise this training so that the players do not feel they are in a ‘boring’ tutorial
mode, but feel situated within the game. Thus there is synergy between “staggered instruction” and sub-domains.

Another feature of game design that exploits the sub-domain design approach is checkpointing and saving. Often, a sub-domain is an episode in a game that requires difficult actions to be performed accurately in a partially-ordered sequence. If the player fails at some point in the episode, the world is reset to the checkpoint, that is the state it was in when the episode began. In other words, the world state is automatically saved as the player begins the episode and restored as many times as necessary until the player successfully completes the assigned task. Often, an episode is divided into multiple “checkpoints” so that partial progress to the goal is also saved automatically. Games have deployed many variations on scaffolding supports for “trial-and-error” learning that could prove useful in an ITS context.

What commercial games typically lack is modeling of the user’s knowledge of the domain. The closest analogue to a student model is the user’s inventory of objects and skills, and the performance history of cleared checkpoints. These are often directly tied to particular sub-domains, so that once a user enters a particular portion of the game, the system knows the user will have a certain items in their inventory and can require those items to be used to make further progress. Unlike ITS student modeling, there is very little focus on individual differences in student knowledge.

Recent research has shown that some of the currently hard-coded learning techniques identified in commercial games can be adapted for control in a model-driven system. In Crystal Island, [71] students are guided through a narrative-centered learning environment to achieve specific learning goals in the domain of microbiology. The ISTS system, [78] adapts its tree-structured dialogue system according to a dynamically updated student model.
Finally, emerging research is uncovering and formalizing underlying game structure. Ludocore [74] had demonstrated that the an event calculus [51] formalism can be used to describe the mechanics of game rules and generate potentially reachable game traces based on those rules. PaSSAGE [82] uses an internal player model to drive the programmatic generation of interactive narrative within a game environment.

2.2 Interactive Narrative and Learning

The field of interactive narrative studies the automatic generation of stories within virtual worlds in which human users interact with one or more computer controlled agents, [43, 40, 76]. Where “interactive narrative” was initially strictly a field of research, the term has evolved to encompass some aspects of digital games, intelligent tutoring, and training simulations. Still, a persistent challenge for interactive narrative is balancing the autonomy of the user, the believability of computer agents, and the coherence of the author’s story-plan. Few systems attempt to strike a dynamic balance between these goals, choosing at design time to either restrict the user’s autonomy and favor a linear story, or provide an abundance of interesting things for the user to do in hopes that the narrative that emerges will be satisfying. This section reviews several systems that have applied interactive narrative to learning: STEVE, the Mission Rehearsal Exercise, and U-DIRECTOR.

2.2.1 Steve: Soar Training Expert for Virtual Environments

Steve [68] is an animated pedagogical agent who teaches human students to operate the engines of a naval surface ship in a virtual environment. Although much of Steve’s contribution derived from its integration of 3D visuals, non-verbal communicative acts, natural language understanding and mixed-initiative instruction, Steve also built on a deep and powerful model
of procedural instruction.

To teach a procedure, Steve requires HTN (Hierarchical Task Network) descriptions of the procedure, including the actions required and the ordering dependencies between actions. Steve’s repertoire of pedagogical operators includes “describe step,” “perform step,” and “explain result” which are combined into communicative suites associated with the hierarchy of action types in the domain. The communicative suites form a class hierarchy that mirrors the HTN, allowing customized extensions via subclassing wherever appropriate. Furthermore, all the operators required for generic conversation with the student (moving, pointing, turn-taking, and discourse grounding acts) are handled independently from these task-based communicative suites.

However, Steve has no student model. Steve’s goal is simply to complete the demonstration of a task. A student has the ability, but is never required, to seize the initiative from Steve and try to perform the next step(s) in the demonstration. Once the task has been completed, the session is ended, regardless of how many steps the student was able to perform. Thus, a student could complete a session having learned an entire procedure or nothing at all.

2.2.2 Mission Rehearsal Exercise

The Mission Rehearsal Exercise [77] extends the model of a pedagogical agent introduced by Steve to a more complex and dynamic environment. This simulation is embedded in an interactive narrative, with multiple non-player controlled agents each with its own set of goals and emotions. Like Steve, however, the content is cleverly presented to give more of an impression of student autonomy than actually exists. The pedagogical content is aligned within a tightly constrained progression through the story.
Although the Mission Rehearsal Exercise is a highly regarded success story within the interactive narrative community, it unfortunately did not extend the depth of Steve’s pedagogical strategies. To quote from one of the earliest MRE papers (“Steve Goes To Bosnia”):

> Our prior work on Steve provides a solid foundation for developing such virtual humans, and our current work is extending Steve in four key areas: a better body, better natural language capabilities, a model of emotions and personality, and more human-like perception [67].

Although both Steve and MRE are properly described as mixed initiative, their worlds are constructed with very low expectations for the level of initiative displayed by the student. A student who tries to do something off the expected path is quickly coerced back onto it. This creates an implicit understanding that the system and its pedagogical agents own the initiative except within narrowly focused windows of delegation. Thus, the student’s exploratory inclinations are throttled.

### 2.2.3 U-DIRECTOR

Mott’s U-DIRECTOR [49] shares Annie’s goal of using interactive narrative to guide exploratory learning. Mott stresses that adapting interactive narrative for exploratory learning demands that user autonomy be sufficient to support exploration of specific “hypothesis-generation-testing cycles.” U-DIRECTOR leverages narrative to produce a broad set of options for intervention, from lighting or sound changes that draw the user’s attention toward a learning opportunity, direct or indirect dialogue supplied by non-player controlled (NPC) characters, and omnipotent environmental control that can be used to dynamically adjust world geography, obstacles, and other challenges.
The key differences between Annie and U-DIRECTOR center around how the two systems perform student modeling and use those models to direct future actions. Mott’s system continuously monitors the student’s performance and uses a Dynamic Decision Network (DDN) to guess what the student is trying to do next. When the student seems to be stuck, a hint is provided in the form of directing the student’s attention toward some area of the world that U-DIRECTOR believes contains information relevant to the student’s likely goal. Annie is more concerned with fixing the particular knowledge gaps revealed in its task-based student model, which may involve multiple hints about different aspects of a particular task in a particular location. Finally, Annie’s interventions are dynamically generated narrative sub-plans that may involve multiple action, whereas those of U-DIRECTOR are primitive actions constructed at design time tied to particular assets in the virtual world.

2.3 Plan-Based Representations and Task-Based Learning

“Task-based,” “task-oriented,” or “procedural” learning is usually suffused into something larger, as was the case with Steve, described in section 2.2.1. Of interest to this research is the extent to which plan-based representations have proven useful to task-based learning. In describing Steve, Rickel [67] pointed out that the plan-based “task representation has proven effective in a wide variety of research on task-oriented collaboration and generation of procedural instructions,” specifically noting Young’s work on plan-based discourse, [93], Mellish and Evans work on using plan structure to generate natural language descriptions [46], and the creation of instructions described by Delin et al. [25]. This section notes additional work employing plan-based representations of task knowledge.
2.3.1 Discourse-focused Tutorial

Since a tutorial session is essentially a discourse between student and tutor, it is reasonable to consider combining a plan-based tutorial representation with the well known plan-based discourse representations. For example, Grosz and Sidner’s [30] SharedPlans formalism has been shown to form a good basis for cooperative communication between two parties (e.g., Rich and Sidner’s COLLAGEN [66] system). A key motivation behind SharedPlans is to distinguish between an agent’s knowledge of a plan schema and that agent’s intention to execute a plan based on that knowledge.

Both SharedPlans and extensions described by Lochbaum [39] evolve the straight-forward STRIPS-style representations into much more expressive and abstract constructions. The additional power of these formalisms would certainly be useful in tutorial generation, leading to pedagogical choices that are more nuanced and perhaps more responsive to student affect.

2.3.2 Agent Learning Approaches

A variety of research has been conducted on the use of plan-based knowledge representations as a basis for agent learning. In most of these systems, the roles between human and agent are reversed from those in ITS: it is a human who is providing the knowledge and the agent that is learning it.

PLOW (Procedure Learning On the Web) [32] extends earlier work on mixed-initiative task-focused dialog to allow humans to teach agents how to perform actions on the web (e.g., purchasing a book). PLOW authors developed AKRL (Abstract Knowledge Representation Language), which builds rules for the execution of hierarchical tasks. Notably, the core of this knowledge representation contains full HTN expressivity.
Garland et al., [27] created a programming-by-example system where the domain expert performs a set of tasks and then annotates each of these tasks to enrich the representation. Blythe adapts the programming-by-example paradigm with the Tailor system [9], for a human to teach an office assistant how to perform computer-based administrative tasks. In KnowMic, van Lent and Laird [85], also use an HTN action representation in a system that enables an agent to learn conditions and goal formations associated with action selection. In contrast to these and similar efforts that largely exploit HTN representations, Lau and Weld focus [37] simple rules on primitive tasks, completely avoiding any representation of task / sub-task relationships.

Unfortunately, applying these agent learning techniques to the tutoring of humans is more complex than simple inversion of roles. Most of these systems are based on a stimulus-response model of learning, where they simply try to encode the optimal production rules to fire when a particular stimulus is observed. What can be taken away, however, is that plan-based knowledge representations have been usefully employed in a variety of human-computer task-centered contexts. The goal for the work in this dissertation is to combine these representations with interactive narrative in a way to provide detailed guidance through exploratory environments.
Chapter 3

Design Overview

A tutor has two fundamental objectives: she wants the student to acquire specific domain knowledge and then develop skill in applying this knowledge to a range of domain-specific problems. From an architectural perspective, a human tutor is a single entity that encapsulates a knowledge base, methods and strategies to transfer that knowledge to the student, and mechanisms to help the student build skills that incorporate the transferred knowledge.

Unlike humans, computer-based tutors are not necessarily equipped with deep and explicit models of the domains they try to teach. In fact, for many effective intelligent tutoring systems, much of the domain knowledge is diffused into their constituent questions, answers, and hints. Much of ITS design is concerned with system-specific algorithms for how and when these various pieces of knowledge are presented to the learner. These algorithms are operating with knowledge that is often at considerable representational distance from the underlying domain. Building tutoring systems on more complete world models should make the teaching algorithms easier to compare and share. However, domain modeling is expensive.
One reason to use a digital game as a learning platform is that it necessarily contains a computable world model of the domain and all the objects contained in the domain (including spatial layout, physics, lighting, sounds, behaviors, and any other attributes the user may apprehend). In other words, a world model comes “for free” as a byproduct of the game development process. What game systems lack are general mechanisms to introduce arbitrary learning objectives and model student progress. Teaching, evaluation and feedback are deeply intertwined with the particular logic and flow of each game.

The goal of this research is to build a generalized tutoring system that can leverage the built-in world model of computer games. This system is named “Annie,” because the task of continuously tutoring in an open non-linear world with uncertain knowledge of the student’s understanding is somewhat reminiscent of the challenges Anne Sullivan faced in teaching Helen Keller the rules of verbal communication. Because Helen was blind and deaf, Anne did not always have the luxury of checking for Helen’s understanding after each new teaching segment. Helen’s emotional issues put constant pressure on Anne to forge ahead in her tutoring, even though her internal model of what was in Helen’s mind became increasingly uncertain. Anne needed to maintain complex theories of just what she thought Helen had learned for several weeks before she was able to reach a famous dramatic breakthrough with Helen. Anne writes:

Helen has taken the second great step in her education. She has learned that everything has a name, and that the manual alphabet is the key to everything she wants to know.

In a previous letter I think I wrote you that “mug” and “milk” had given Helen more trouble than all the rest. She confused the nouns with the verb “drink.” She didn’t know the word for “drink,” but went through the pantomime of drinking whenever she spelled “mug” or “milk.” This morning, while she was washing, she wanted
to know the name for “water.” When she wants to know the name of anything, she points to it and pats my hand. I spelled “w-a-t-e-r” and thought no more about it until after breakfast. Then it occurred to me that with the help of this new word I might succeed in straightening out the “mug-milk” difficulty. We went out to the pump-house, and I made Helen hold her mug under the spout while I pumped. As the cold water gushed forth, filling the mug, I spelled “w-a-t-e-r” in Helen’s free hand. The word coming so close upon the sensation of cold water rushing over her hand seemed to startle her. She dropped the mug and stood as one transfixed. A new light came into her face. She spelled “water” several times [35].

To enable this momentous learning experience, Anne had continued expanding a complex student model in the face of increasing uncertainty of its fit with Helen’s mental model. It was necessary for Helen to acquire a base of incomplete and uncertain knowledge before any of it could “click into place.” Similarly, a tutor must allow a student working within a digital game environment to make forward progress, even when it means the tutor lessens confidence in its student model. Technically, a system could interrupt gameplay constantly to check for understanding, but in practice users resent the distraction of continual interrogation or instruction (e.g., Microsoft’s “Clippy,”[54] or Legend of Zelda’s “Navi”[41]). In order to adapt the digital game paradigm for teaching, Annie must be capable of working several steps ahead of a student model whose accuracy cannot be guaranteed.

### 3.1 System Architecture

To accommodate the likely lag between student action and its interpretation, Annie maintains independent representations of the **world model** and the **student model** as shown in figure
3.1. The **world model** describes what the virtual world contains and how it works. The student forms an internal mental model of this world, initially empty, but illuminated and expanded as the student explores the world. The student’s “avatar” is included in the thought bubble of figure 3.1 because in games where the student has an avatar, it will play a large role in the structure of the mental model. The **student model** is Annie’s representation of the student’s mental model, based on observations of the student in the virtual world. The **tutorial director** matches domain-specific learning objectives and teaching strategies against perceived gaps in
the student model. Note the contrast between this and traditional game-based learning, where the student model and the teaching strategies are deeply and implicitly embedded in the structure of the world. Annie promotes much better context sensitivity, extensibility, customization and reuse of teaching strategies through a domain-independent architecture.

Figure 3.1 also depicts Annie’s dependency upon the Zócalo system [88], which is the service-oriented architecture implementation of Mimesis [69]. Zócalo contains a decompositional partial order planner that generates story plans that execute in a virtual world. The Execution Manager component of Zócalo controls the progression of this story plan, interceding between the student and the game world as required to ensure that story goals are realized. To generate plans, Zócalo is provided with a library of plan operators and a description of the initial and desired goal states. As a consequence of the generating the story plan, Zócalo builds a plan space that contains incomplete plans, as well as plans which successfully reach the goal conditions. Annie is responsible for maintaining the plan-based description of the operators, objects, and conditions associated with the Zócalo game world. To aid discussion, we refer to the Zócalo game world as the ZWorld, and the library of plan-based descriptions maintained by Annie as the ZLib.

Annie’s direct connection to the Zócalo execution manager allows it to observe all student action attempts and outcomes, supporting Annie’s mission to teach students task-based learning: essentially how to make things happen in the world. A focus on task-based learning narrows considerably the scope of teaching. First, it means that Annie teaches a thin, but important, slice of the world model that describes the possible tasks, or actions that can happen in the world. This slice of the ZWorld model is exactly the set of planning operators contained the ZLib. Second, the student does not need to learn every detail of every operator in the
library, but only that portion that is required to accomplish the tasks at hand. Annie can mine the analytical insight of plan space planning to prioritize exactly this necessary subset of the operator library the student needs to know.

### 3.2 Operational Overview

Figure 3.1 provides a system-level view of Annie’s relationships with other components of the system. Specifically, it shows that Annie maintains and analyzes several data models to inform its interactions with the execution manager. What is not obvious from the figure is that a big chunk of Annie’s work is the creation of these models that occurs during initialization. After the models are initialized, Annie enters into an execution loop that iterates over tasks in the ZWorld represented by plan structures.

#### 3.2.1 System Initialization

Before Annie begins a tutorial session with the student it must perform two important initialization tasks. First, Annie constructs its knowledge base, called ABase. The ABase is initialized with all the information in the ZLib. The second main initialization task for Annie is to generate an initial tutorial plan.

For planning, Annie uses the Longbow [95] decompositional partial order planner found at the core of Zócalo. Through Zócalo, Annie creates partially ordered plans consisting of partially-ordered sequences of operators that can transition the world from its the initial state to the goal state. If Annie were presented with a planning problem that did not specify any goals pertaining to student beliefs, a story plan would be produced consisting of a partially-ordered sequence of actions that achieves a particular goal state, where some of these actions
will be taken by system actors (NPCs), and others by the student, just as would be the case with the Zócalo and Mimesis systems. As student beliefs are added to the goal state, however, the need arises to evolve the story plan into a tutorial plan. A **tutorial plan** is one that not only achieves a particular goal state in the world, but also achieves a particular state in the student model. In a tutorial plan, it is not enough for the student to perform a certain set of actions, the student must also demonstrate particular understanding of those actions.

An important distinction must be made here. Annie builds tutorial plans by *elaborating* the story plans produced by Longbow with additional actions designed to bring about changes in the student model. The distinction is that Longbow remains unaware of both the conditions comprising the possible states of the student model and the actions designed to change those states. This process of plan elaboration begins at initialization, and continues throughout execution, as Annie observes the student’s behavior and updates the student model.

Thus, the final step of Annie’s initialization is creating a tutorial plan consisting of a plausible partially-ordered sequence of student and system-initiated actions that is guaranteed to bring about a specific goal state for the world. Annie uses this plan to initiate the tutorial session within the ZWorld. The tutorial plan cannot guarantee that the goal beliefs for the student will be achieved. Annie merely monitors student behavior and tries to optimize the frequency and extent of its tutorial interventions to increase the likelihood that the goal beliefs are acquired. The details of this process are presented in section 4.1.

### 3.2.2 Execution Cycle: Overview

Armed with a richly detailed $ABase$, an initial tutorial plan and a space of alternative plans, Annie can begin the learning session. Like a human tutor, Annie should be extremely atten-
tive to the actions of the student, deferential to student initiative, but skillful and clever when intervention is warranted. However, unlike a human tutor, Annie can exploit the learning environment to intervene with much greater subtlety. For a human tutor it is often difficult to identify an intervention that provides the minimal required help. Often, a human-tutored student is able to derive more useful meaning from tutorial interaction than might be desired. Annie can disguise or hide pedagogical intent through interventions that make changes in the virtual world and multiple non-player controlled agents.

Annie’s interventions can be as simple as increasing or decreasing the difficulty of the next task based on the student’s speed and accuracy on the current task. More complex are Annie’s interventions to help the student complete, with understanding, a partially executed task. Annie uses the student’s action history and the continually-updated $A_{Base}$ to recognize when intervention is needed, choose from alternative possible remedies, and choose between alternative methods for expressing those remedies.

The structure of Annie’s run-time behavior consists of a cycle, or loop. This loop iterates continuously, with a nominal delay to conserve computing resources. Each time an action is taken, either by the student or the system, Annie updates plan history and the student model. Annie then consults these data to decide if intervention is warranted, resulting in either a system-based action, or an anticipated student action.

Execution loops are common in Intelligent Tutoring Systems. VanLehn has suggested that many of the seemingly dissimilar tutoring systems [86] that comprise the core of ITS research, use a common execution model consisting of a pair of nested loops. In this model, an outer loop iterates over “tasks” and an inner loop that iterates over each step in a task. Because Annie’s plans contain hierarchical decompositions, its single loop is sometimes iterating over tasks and
sometimes over steps in a task. Unlike a concentric loop architecture, however, Annie is free to switch to a completely different task as required.

As depicted in figure 3.2, Annie’s execution cycle is divided into five stages.
Stage One: Remediation Consideration  In this stage, Annie considers whether to continue with the existing tutorial plan, or to introduce a remediation, or change the plan. Annie weighs several factors in making this determination. First, Annie reviews the student model for unrealized pedagogical goals. It then compares the beliefs constituent to proximal tasks in the tutorial plan to identify misconceptions that may hinder the student’s immediate progress. An urgency threshold for repairing these misconceptions is derived based on the proportion of steps remaining in the plan. If the threshold is met, the most critical misconception is chosen for remediation. To integrate a remediation into the tutorial plan, Annie proceeds to stage two. Otherwise, if no candidate remediation is deemed sufficiently urgent, Annie proceeds directly to stage three.

Stage Two: Plan Revision  Annie selects from several different abstract plan templates with to remediate the misconception indicated in stage one. Domain-independent templates are provided that correspond to common ITS interventions including prompt, hint, demonstrate, teach or do. In addition, Annie can select from a library hand-authored alternative sub-plans particularly attuned to common misconceptions for a particular domain. As a last resort, Annie may abandon the currently active tutorial plan in favor of another potentially successful plan in the plan space, if it offers a better fit for the current plan history. Remember that Annie is largely a spectator, and its active plan is merely a guess at a likely course of action for the student. If the student’s actions show a pattern of increasing divergence from the current plan, coupled with increasing similarity to an alternative plan, it is rational for Annie to convert to that representation. Once a remedy or alternative plan is selected, a sub-plan particular to the current plan context is generated, and the existing tutorial plan is repaired to include this new sub-plan.

Stage Three: Action Specification  Annie enters stage three with a valid plan, from which it
must single out the action to be performed on this iteration through the execution loop. There may be more than one potential action that could be executed next, and these actions may be either student or system-initiated. Annie selects the next action based on its projected pedagogical utility.

**Stage Four: Execution** This is the simplest stage of the loop. Annie sends a message to the execution manager telling it the next action to be executed.

**Stage Five: Action Result Processing** The plan history, current state of the ZWorld and the student model are updated based on the action result. An extensive catalogue of action result scenarios, guides Annie in decided how to update each component of the student model.

The obvious two-fold challenge for Annie is to acquire the student’s knowledge of the operators in the library, and then leverage this knowledge to improve tutorial planning. As the student progresses, Annie gains more and more information about the state of the student’s knowledge, but has less and less time remaining to act on these revelations. This poses a difficult problem, but it is problem that demands only a reasonable solution, not a perfect one.

The goal is for Annie to model the behavior of a human tutor. It should be extremely attentive to the actions of the student, deferential to student initiative, but skillful and clever when intervention is warranted. However, unlike a human tutor, Annie can exploit the learning environment to intervene with much greater subtlety. For a human tutor it is often difficult to intervene without providing more help than necessary. Often, the student is able to derive more useful meaning from tutorial interaction than the tutor intends. Because Annie can deliver interventions through changes in the virtual world and through multiple non-player controlled agents, pedagogical intent can be disguised and unintended consequences can be minimized.
Chapter 4

Detailed Design

This chapter describes, in detail, the design of the current implementation of the Annie system. First, the set of inputs required by Annie are described in detail. Next, the five stage execution cycle is described sequentially, from stage one through stage five. The penultimate section of this chapter provides more detail on the diagnostic mechanisms employed in stage five, and the final section is a brief summary.

The execution cycle iterates twice per second, so Annie can respond quickly to new information when appropriate. However, if Annie were to interrupt the student’s exploration of the world on each iteration, or even every tenth iteration through the loop, the student would become annoyed, or frustrated and would likely stop learning. Instead, the system tends to react on those iterations through the loop that happen to come immediately after receiving notification of an action having been executed in the learning environment. Thus, when reading sequentially through the steps in the execution cycle, it is a good idea to imagine that such a notification has recently been received and processed in stage five of a previous iteration.
4.1 Annie’s Inputs

The knowledge representation required as input by the Annie system is similar to that required by other systems that perform automated planning. Typical automated planning systems [55, 44, 33] require as input a planning problem description, often referred to more simply as a planning problem, and typically expressed through the STRIPS [26] language, or one of its variants. Annie requires as input a planning problem expressed in a STRIPS-like language and, in addition, it requires a description of the learning objectives Annie should try to achieve during the learning session. In Annie, these learning objectives are restricted to describing a student’s knowledge of the conditions in the ZWorld (as described earlier, the phrase ZWorld is the term used by the Zócalo service-oriented architecture, on which Annie is built, to describe the learning environment). Specifically, Annie is concerned with the student’s knowledge of conditions in the ZWorld that are related to each action’s successful execution (i.e., their preconditions and effects, as described below).

Annie’s planning problem is referred to as the Learning Problem Description, or LPD.

Definition The Learning Problem Description (LPD) is a 4-tuple \( \langle I, G, ZLib, LO \rangle \), where \( I \) describes the initial state of the learning environment, \( G \) is a partial description of the goal state of the ZWorld, and \( LO \) is the set of Learning Objectives for the student that Annie strives to establish.

The initial state \( I \) is represented by a conjunction of function-free literals describing all the conditions that are true in the ZWorld when the plan begins to execute. When producing tutorial plans, Annie uses the Closed World Assumption [65]: any condition not specified in the initial state is assumed to be false.
Like the initial state description $\mathcal{I}$, the goal state description $\mathcal{G}$ is represented by a conjunction of function-free literals. $\mathcal{G}$ represents those conditions in the $\text{ZWorld}$ that must hold at the completion of a successful plan’s execution.

The $\text{Zócalo Operator Library}$, or $\text{ZLib}$, describes a set of operators Annie can use to construct tutorial plans. Each operator consists of a unique name, a conjunction of function-free literals called preconditions that describe what must be true immediately before the action can be executed, and a conjunction of function-free literals called effects describing state changes brought about by execution of the action. Because Annie reasons about a student’s beliefs relative to an operator’s preconditions and effects, each precondition or effect of an operator is required to have unique identifier called its name. One method to maintain both readability and uniqueness of these names is to apply a pre-processing step to the $\text{ZLib}$ that uses a global counter to generate a unique integer for each precondition and effect of each operator in the $\text{LPD}$ and append this integer to the relation symbol corresponding to each precondition or effect. For example, a name of “hasTool-17” might be generated to ensure uniqueness of the identifier for the “hasTool” effect of operator “PlayerMoveToPickupNiceTool”, if at that point in the pre-processing, 16 names had already been generated.

In addition to the preconditions and effects lists common to many planning representations, Annie also supports the representation of an operator’s constraints, a conjunction of function-free literals that represent, like preconditions, conditions that must hold immediately before an action executes. Unlike preconditions, however, constraints are conditions that cannot be planned for. That is, they do not appear in the effect of any action operator. Constraints are generally used to act as filter conditions [34], that is, they aid planner efficiency by reducing the size of the set of ground terms to be evaluated as bindings for terms specified in the pre-
conditions and effects of the operator. The terms or parameters of an operator are implicitly defined as the disjunction of all the object variable symbols found in any of the preconditions, effects, or constraints of the operator.

Also, Annie requires two additional kinds of data to be defined for each operator. First, a boolean value bStudentInitiated must be associated with each operator, where a value of true means that the operator is performed by the student and a value of false means the operator is performed by the system. Second, a text field operatorDescription must be associated with each operator and contain template text to be used by Annie to generate help text describing the purpose of the operator. Similarly, each of the conditions included in any of the preconditions or effects lists for any operator must also have an associated template text field conditionDescription containing text to be used by Annie to generate help text describing that condition.

The other component of the LPD is \( \mathcal{LO} \), the set of student learning objectives Annie works to establish. In order to give a precise definition of a learning objective, some additional definitions are first required. Annie targets task-based learning, which means that its learning objectives are restricted to facts about the preconditions and effects of operators in the ZLib. To underscore that the items Annie endeavors to teach all have this specific focus, each such fact is described using the Annie-specific term: ZFact.

**Definition** A ZFact is a triple \((\text{operatorName}, \text{conditionExpression}, \text{conditionName})\), where operatorName is the name of an operator opr in the ZLib, conditionExpression is either “hasPrecondition”, or “hasEffect”, and conditionName is the name of a precondition or an effect of opr.
Annie cannot know with absolute certainty what is in the mind of the student. Instead, Annie estimates the likelihood that the student believes, or knows, a particular ZFact. In the current implementation, these estimates are represented using a coarse model of five discrete levels of belief likelihood: “HighlyLikely”, “Likely”, “Neutral”, “Unlikely”, or “HighlyUnlikely.” Future work will consider replacing this coarse model of student belief likelihood with a more precise Bayesian Belief Network [18], or similar model, where the range of likelihood estimates is continuous. The combination of a ZFact with a belief likelihood estimate is referred to with the term ZBel. Table 4.1 describes the semantics of a set of example ZBel tuples.

**Definition** A ZBel is a tuple \(\langle\text{statement}, \text{beliefLikelihoodEstimate}\rangle\), where statement is a ZFact, and beliefLikelihoodEstimate is one of: “HLB”, “LB”, “NB”, “UB”, or “HUB”.

### Table 4.1: Belief Strength Levels

<table>
<thead>
<tr>
<th>Level</th>
<th>Example ZBel Tuple</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highly Likely</td>
<td>(\langle\text{opr1, hasPrecondition, } P_2, \text{HLB}\rangle)</td>
<td>It is highly likely the student believes (\text{opr1}) has precondition (P_2)</td>
</tr>
<tr>
<td>Likely</td>
<td>(\langle\text{opr1, hasEffect, } P_4, \text{LB}\rangle)</td>
<td>It is likely the student believes (\text{opr1}) has effect (P_4)</td>
</tr>
<tr>
<td>Neutral</td>
<td>(\langle\text{opr1, hasEffect, } P_6, \text{NB}\rangle)</td>
<td>It is neither likely nor unlikely the student believes (\text{opr3}) has effect (P_6)</td>
</tr>
<tr>
<td>Unlikely</td>
<td>(\langle\text{opr4, hasPrecondition, } P_8, \text{UB}\rangle)</td>
<td>It is unlikely the student believes (\text{opr5}) has precondition (P_8)</td>
</tr>
<tr>
<td>Highly Unlikely</td>
<td>(\langle\text{opr5, hasEffect, } P_8, \text{HUB}\rangle)</td>
<td>It is highly unlikely the student believes (\text{opr5}) has effect (P_8)</td>
</tr>
</tbody>
</table>
Within Annie, a total ordering is defined over belief likelihood estimates reflecting the relative strength of the Annie’s estimate of the likelihood that the student knows a given ZFact. Specifically $HUB < UB < NB < LB < HLB$. When two belief likelihood estimates $B_0$ and $B_1$ differ such that $B_0 < B_1$, one can say that $B_1$ reflects a stronger belief than $B_0$ and $B_0$ reflects a weaker belief than $B_1$. Likewise when a particular ZBel changes such that the new belief likelihood estimate is greater than the previous value, this can be described as an increase in the belief likelihood, where the opposite would be called a decrease.

Each learning objective $l$ in $LO$ is a ZBel of the form $\langle ZFact, beliefLikelihoodEstimate \rangle$. During a tutorial session, Annie attempts to establish for each learning objective, through its monitoring of student progress along tutorial plans, an estimate of belief likelihood for the ZFact of $l$ that meets or exceeds the specified belief likelihood estimate. An alternative approach would be for Annie to consider negative beliefs (“Unlikely”, or “HighlyUnlikely”) as satisfied when belief estimates are equal or lower than the targets. This alternative approach was not chosen for the existing implementation.

An example of a learning objective that could be included in $LO$ for the FixIt learning environment is shown below:

$\langle \langle \text{PlayerMoveToPickupNiceTool, hasPrecondition, isToolSpawnLoc-42} \rangle, LB \rangle$

This learning objective would be satisfied when it is estimated to be “Likely” or “Highly-Likely” that the student believes that operator “PlayerMoveToPickupNiceTool” has a precondition named isToolSpawnLoc. Section 4.3.5, which concerns the processing of action results received from the learning environment, describes how Annie assigns and maintains belief levels within its student model. The next section shows that Annie’s student model is built with
the same $ZBel$ constructs that comprise the $LO$.

### 4.2 System Initialization

Annie takes as input the $LPD$, described above in section 4.1. Upon input of the $LPD$, Annie begins to initialize its knowledge base, called the $ABase$. The $ABase$ contains all the data structures beyond the $LPD$ that Annie uses to perform its tutorial reasoning.

**Definition** The $ABase$ is a 5-tuple $\langle SM, PPSS, ET, DT, RT \rangle$, where $SM$ describes a student model, $PPSS$ is a planning problem solution set, $ET$ is an execution trace of plan actions, $DT$ is a set of diagnostic templates and $RT$ is a set of remediation templates.

The student model $SM$ holds Annie’s model of the student’s current beliefs about the operators available for actions in the $ZWorld$, represented as a set of $ZBels$. To begin initialization of the $ABase$, $SM$ is initialized with one $ZBel$ tuple for each precondition and effect of each operator in the $ZLib$. In the current implementation, the initial default belief level applied to each of these conditions is “NB”, or neutral. Future work could examine the provision of input files to describe knowledge specific to a particular student or lesson.

The $PPSS$ holds a set of solutions to the planning problem described by the $LPD$. Unlike many plan-based systems, Annie cannot rely on an single solution plan for all of its reasoning, because the student is not guaranteed to be following any particular plan. Rather, Annie monitors multiple success paths that the student may take to reach the goal state. To initialize the $PPSS$, Annie extracts the planning problem components ($I$, $G$ and $ZLib$) from the $LPD$ and sends them as a planning problem to the planner through the $Zócalo$ service-oriented architecture. The planner returns a set of solution plans, which are stored in the $PPSS$ component.
of the ABase. Each of these plans represents a successful solution to the current tutorial task whose actions could be considered for execution by the student. The plan with the fewest steps of all the plans in PPSS is designated as the current plan $P_c$, indicating that this is the plan Annie assumes is currently being followed by the student. Although more complex heuristics could be used, the current implementation of Annie chooses the plan with the fewest actions as the initial $P_c$. There currently is no model or theory determining the best length of tutorial plans for ITS systems, but other systems [69, 15, 49] opt for the same economy of steps approach taken here. In terms of the plan modeling a user’s expected behavior, a shortest-plan approach has the benefit of providing Annie as a starting point an optimal path onto which additional remedial actions may need to be placed.

It is important to note that the plans do not ensure the achievement of learning objectives. These are addressed by the remediation actions produced by algorithms described in section 4.3.2.

The $ET$ component of ABase is used to track a student’s progress relative to the set of solution plans in PPSS. As there can be no guarantee that a student is following a particular plan from start to finish, when each action executes in the ZWorld, Annie updates the traces of all solution plans that are being tracked. Thus, $ET$ contains a set of execution traces, one for each plan in the PPSS. Several data structures comprise an execution trace. First, the execution trace contains the original plan description, which is a set of actions and ordering constraints between them. Second, a modifiable directed acyclic graph (or DAG) is built from the original plan description to track execution of the plan. Initially, each of the actions in the original plan is a node in the DAG and each of the ordering constraints creates an edge between nodes. As actions are executed, they are removed from the DAG. This provides the $ET$ with

49
a convenient partitioning of the set of actions of the plan into two subsets: the set of actions that have been executed and the set that have not. Annie updates the DAG and this partitioning upon each notification of action execution received from the learning environment.

The $\mathcal{DT}$ component of $ABase$ contains the diagnostic templates Annie uses to identify particular student misconceptions. These are currently hard-coded components of the $ABase$, but the intention is for future implementations of Annie to initialize $\mathcal{DT}$ by loading these templates from external files. The remediation templates are described at length in section 4.3.5.

The $\mathcal{RT}$ component of $ABase$ contains the remediation templates Annie uses to choose the most appropriate scaffolding to use to remedy particular student misconception. The remediation templates are described at length in section 4.3.2. As with the diagnostic templates, these templates are hard-coded components of the $\mathcal{RT}$ component of the $ABase$, but the intention is for future implementations of Annie to initialize $\mathcal{RT}$ by loading these templates from external files.

### 4.3 Execution Cycle

As described in section 3.2, once the system is initialized, Annie establishes communication with the learning environment, and then begins to iterate through its execution loop. The loop iterates continuously, with a nominal wait inserted (one half second in the current implementation) to avoid wasting computing resources. On each iteration, the loop proceeds through five stages. The key contribution of each stage is described in the following list for reference:

**Stage One: Remediation Consideration**  This stage reviews the current state of student beliefs to decide if the student has a misconception sufficiently urgent as to warrant reme-
Stage Two: Remediation Selection  This stage uses the remediation templates to select the most appropriate remedial action for Annie to take in the learning environment, based on the particular misconception selected in stage one. If no misconception was selected in stage one, this stage is bypassed.

Stage Three: Action Specification  This stage chooses the next action to be executed in the ZWorld, which may be a remedial action selected in stage two, or another action in the current plan.

Stage Four: Execution  If an action is selected in stage three, a command directing its execution is sent to the learning environment.

Stage Five: Action Result Processing  This stage checks the buffers Annie uses for notifications of action successes and failures to see if any ZWorld action results have been reported since the last iteration. It then uses the diagnostic templates to determine which, if any, student beliefs should be altered as a result of the action results.

4.3.1 Stage One: Remediation Consideration

Stage Overview  In this stage, Annie considers whether to continue with the current tutorial plan, $P_c$, or to introduce a remediation, or change, to the plan. Annie weighs several factors in making this determination. First, Annie compares the student model $SM$ to the learning objectives $LO$ to identify any learning objectives that are as yet unrealized. It then determines the proximal tasks by referring to the execution traces in $ET$. A proximal task is an action that either can be executed immediately, or executed after at most two additional tasks execute. Annie surveys the student’s beliefs about the preconditions and effects of the operators associated
with proximal tasks in the ET to identify misconceptions that may hinder the student’s immediate progress. An urgency threshold for repairing these misconceptions is computed based on the remaining steps in the plan. If the threshold is met, the most critical misconceptions is chosen for remediation. To integrate a remediation into the tutorial plan, Annie proceeds to stage two. Otherwise, if no candidate remediation is deemed sufficiently urgent, Annie proceeds directly to stage three. The methods Annie uses to compute the set of proximal actions is described below.

**Stage Details** Annie continually monitors the steps in the current plan, $P_c$, that are nearing execution, while also considering opportunities to achieve LPD learning objectives and guide the student toward successful plan sequences. Annie continually examines the set of proximal actions, those nearest to being executed, for student “misconceptions.” A misconception is a $ZBel\ zb$ in $SM$, of the form $\langle ZFact, beliefLikelihoodEstimate \rangle$, where there is a deficit between the current belief likelihood estimate contained in $zb$ and the target level of belief Annie would like to achieve. This can happen for one of two reasons. First, the current belief for $zb$ may be below the target described by $LO$, the Learning Objectives component of the LPD. Second, if the operator component of the $ZFact$ in $zb$ is associated with a proximal action whose current belief level is lower than “NeutralBelief” then $zb$ is marked as a misconception. Alternative approaches could include identifying the most critical actions, or considering models of problem solving that include chains of actions. The current implementation focuses on proximal actions because they are the most likely to be connected to what the student is currently doing, so the scaffolding Annie provides based on those actions is more likely to be put to immediate use.
Remediation Consideration Steps  The sequence of steps Annie proceeds through in considering which, if any, misconceptions require remediation is:

1. **Identify and rank LPD misconception set.** The LPD misconception set (LPDMS) is the subset of the LPD beliefs that are not currently satisfied, meaning that the current state of that belief in the student model is at a lower belief strength than what is specified in the LPD goal state for that belief. Annie ranks each misconception according to the number of belief levels separating its current belief strength and the goal belief strength given in the LPD.

2. **Identify and rank proximal action misconception set.** The proximal action misconception set (PAMS) is the subset of the beliefs about the proximal actions in the set of possibly successful plans whose current value in the student model is set below “Neutral Belief.” The first step in creating this set is to categorize these actions according to how close they are to being executable. For example, figure 4.1 depicts a situation where action B has completed successfully. The lower case letters in the figure represent preconditions and effects, upper case letters represent actions. As defined above, the proximal set of actions contains those filled in with vertical lines (\{C, D, E, and G\} in figure 4.1. Any of those actions could be executed next, as their preconditions are currently satisfied, i.e., zero intermediate steps are required before they can execute. This group of actions is labeled “Tier 0.” The proximal set also contains another tier of actions (Tier 1 = \{H, F\}: faint horizontal lines), those whose preconditions can be satisfied by a causal chain of length one. Finally, the proximal set contains a third tier, those whose preconditions are satisfied by a causal chain whose minimum length is 2. For example,
action K is in tier 2. To avoid confusing the student with concepts beyond the visible horizon, Annie does not extend its search beyond tier 2.

Annie partitions the actions in all the plans in the $\mathcal{PPSS}$ into these tiers by analyzing the $ET$ component of the $A\text{Base}$. In cases where a given action appears in more than one tier, it is labeled with the lowest tier in which the action appears. The proximal action belief misconception set is then created by collecting the misconceptions associated with
the actions in a given tier and placing them in a corresponding tier of the proximal action misconception set.

3. **Calculate the mean goal proximity ratio (MGPR).** The MGPR is a rough measure of the percentage of plan actions that remain before the goal state is achieved. Annie uses the MGPR when determining the urgency with which Annie guides a student toward known paths to success. For a given plan in \( \mathcal{PPSS} \), the goal proximity ratio (GPR) is calculated by first counting the number of unexecuted actions shown in the \( \mathcal{ET} \) for the plan, \(|A_{\text{unexecuted}}|\), and dividing that number by the total number of executable actions in the plan.

\[
GPR = \frac{|A_{\text{unexecuted}}|}{|A|}
\]

The MGPR is the mean GPR for all plans in \( \mathcal{PPSS} \).

4. **Select a misconception that exceeds a threshold.**

The process for determining when to select a particular misconception for remediation is detailed below. This process is admittedly ad-hoc and will be revised and extended as Annie is applied to more domains and games. Future work will make the process more easily modifiable by plan authors (for instance, via use of a graphical user interface and a rule-based specification language) as it may be necessary to set them differently in different environments.

(a) Annie not only monitors system-executed actions, but actions in its plans that are to be executed by the student. When these actions fail to be executed in a timely fashion, it may mean that the student has incorrect beliefs about their preconditions or effects. The process for tracking these actions and identifying these misconceptions is described in section 4.4.1. Let \( M_a \) be the set of those misconceptions in
Tier 0 of the proximal action misconception set (PAMS) that arise when a student fails to execute a planned action in a timely fashion. If $M_a$ is not empty it means that one or more actions the student should be taking are not being taken. In this case, non-deterministically select some misconception $m_a$ in $M_a$ for remediation.

(b) If no action is chosen in step (a), let $M_b$ be the intersection of PAMS and LPDMS. If $M_b$ is not empty, rank the beliefs in $M_b$ according to the ranking function of the LPDMS, that is, the difference between current belief strength and target belief strength and non-deterministically select some element $m_b$ of $M_b$ for remediation.

(c) If neither of the previous criteria are met, Annie proceeds to evaluate the PAMS by itself. Let $M_c = PAMS$.

- When the MGPR $> 0.7$ (most actions have yet to be executed), remove from $M_c$ all beliefs misconceptions not in tiers 0, 1 or 2 as well as all misconceptions with $HUB$ (highly-unlikely) beliefs.
- When the MGPR $< 0.7$ and MGPR $> 0.3$, remove from $M_c$ all misconceptions with belief levels not of $UB$ or $HUB$ in any tier.
- When MGPR $< 0.3$, remove all misconceptions from $M_c$ not in tier 1 or not with belief levels of $UB$ or $HUB$ (as many of the alternative branches may not come to pass).

If $M_c$ is not empty, let $n$ be the lowest tier for which misconceptions appear in $M_c$. Rank all misconceptions in $M_c$ in tier $n$ from high to low according to differences between current belief levels and targeted belief levels. Non-deterministically select for remediation a misconception $m_c$ from those most highly ranked in $M_c$.

(d) If no misconception has yet been selected and MGPR $< 0.3$ (that is, there are few actions remaining to be executed), non-deterministically select a misconception...
from LPDMS for remediation.

(e) If no misconception has yet been selected, return no misconception for remediation.

The threshold values described in the previous section (e.g. “MGPR > 0.7”) are merely arbitrary parameters derived independent of any particular theories of learning or plan reasoning. The purpose of these thresholds is to restrict the frequency with which Annie will attempt remediation. While these values worked well enough for the main evaluation (described below in section 5.4), as future work exposes Annie to more different games and domains, it is likely that this scheme will evolve. For instance, the process of choosing both what and when to remediate could also consider task difficulty and general student aptitude.

Given that Annie is looping through its execution cycle twice per second, it will most often be the case that, upon entry into the loop, no candidate remediation exceeds the urgency threshold and Annie will proceed directly to stage three (action specification). Only when a misconception is chosen for remediation will Annie proceed to stage two (remediation selection) to revise the plan. There are two key reasons for Annie’s remediation reticence. First, the student needs to be in charge. Annie’s interruptions will become increasingly transparent and annoying as they break the student’s flow and sense of being in charge of the learning experience. Second, Annie must respect the uncertainty inherent in its projection of the student’s plan. Annie runs the risk of remediating misconceptions in actions that are not even required for the student’s plan to succeed. This could result in Annie oscillating between several plans as the student adjusts to Annie’s actions.
4.3.2 Stage Two: Remediation Selection (when remediation is indicated)

The purpose of stage two is to identify the plan alteration required to address the misconception identified in stage one. In the most general case, Annie may discover it is necessary to generate a new sub-plan, and repair the existing tutorial plan with this new sub-plan. For the current implementation of Annie, the range of possible remediations is constrained so that none are so extensive as to require plan repair. Rather, all remediations consist only of actions that do not change the state of any learning environment. Specifically, all the actions constituent to remediations in the current implementation are restricted to communicative acts to be performed in the learning environment.

Remediation Templates - Overview

Annie addresses to the learning needs of the student by selecting the most appropriate available remedial template to generate scaffolding for the identified misconception. As described in section 4.3.5, each misconception is identified based on Annie’s identification of the appropriate diagnostic template in effect upon notification of a particular action result. The assigned diagnostic template remains associated with the misconception in the $SM$, and is used in this remediation phase to tailor system action. Thus, this section of the document uses the same taxonomy of scenarios used in the diagnoses of misconceptions.

The remediation templates target beliefs about preconditions or about effects; templates describe a range of remediations analogous to the “prompt, hint, teach, do” of mainstream ITS. These scaffolds make use of the template text provided in the operator library $Zlib$ for each operator and each condition in the domain (as described in section 4.1). In the template examples provided below, the function $getDescription(opr)$ returns the text $operatorDescription$ of an operator that Annie uses to customize the text used by the template to communication to
the student. Similarly, the same function applied to precondition or effect names returns the
*conditionDescription* text description for the corresponding condition.

Where the outcome of a particular remediation is to convey a text message to the student, this
is accomplished in the current implementation by sending a message to the learning environ-
ment containing the text. FixIt extracts the text from the message and employs a text-to-speech
system that provides an audible announcement to the learner. In addition, the text of the mes-
sage is displayed on the on-screen HUD, or heads-up display.

Although the intention is for Annie to eventually load remedial templates from external files,
they are currently hard-coded and limited by the functionality provided by the learning envi-
ronment. For example, where some of the templates described in this section specify that a
non-player controlled (NPC) character take a particular action, or deliver a particular message,
the FixIt learning environment does not support the direction of NPCs, so these remediations
were not available in the formal evaluation. The descriptions are retained here to provide an
idea of future possibilities for Annie’s development. The focus of the current implementa-
tion was to replicate the text-based scaffolding traditionally employed in ITS, rather than imme-
diately exploiting a broad range of of untested scaffolding strategies employing NPCs. For
completeness, there are references within the “Show” and “Do” templates for possible remedi-
ations. As an optimization, however, a field in ZLib is used to mark the actions for which these
remediations have corresponding pre-defined scripts in the learning environment. Where such
scripts do not exist, Annie does not choose those remediations.
Remediation Templates - Selection Reasoning

When more than one remedial template is found that corresponds to a particular misconception, Annie uses a simplified selection algorithm similar to the “Fixed Strategy” described by Murray and vanLehn [52]. In the “Fixed Strategy,” a sequence of progressively more assertive remedial actions are executed until the targeted task is completed. This strategy escalates from an initial remedial action of “Prompt” to “Hint” to “Teach” to “Do.” The current implementation of Annie selects remediations according to the following, admittedly unsophisticated, scheme. The first time that a remediation is selected for a particular operator, a “Prompt” remediation is chosen if one is available. Otherwise, a “Hint” is chosen. The second time a remediation is selected for a particular operator, the “Prompt” remediation is reselected only if no “Hint” is available for the misconception, otherwise “Hint” is selected. On the third remediation for a given operator, the first template found in the sequence “Show”, “Teach”, or “Do” will be performed. After the third iteration of remediations for a particular operator, Annie will no longer select ZBels associated with that operator for remediation. Future work may compare effectiveness of this scheme with a decision-theoretic remediation selection algorithm.

Remediation Scenario 1A: Student Inaction - Unknown Precondition

This remediation scenario arises when Annie decides in stage one to attempt to raise the student’s level of belief that a particular operator has a particular precondition, where the most recent reduction in that belief was a result of diagnostic scenario 1A (described in section 4.4.1 on page 78). Diagnostic scenario 1A arises when the student fails to take an action that is needed before the rest of the plan can continue.

For example, in the FixIt domain, it may be the case that the student has failed to fix a runaway process by applying the “nice” tool, as the student is unaware that a precondition of
using the “nice” tool is that is must be manually selected, that is, that the tool must currently be in the hand of the player’s character. This might be reflected in the student model with the following statement:

\[
\langle \text{PlayerFixRunawayProcess, hasPrecondition, isSelectedNiceTool-42}, \text{HUB} \rangle
\]

For the templates below, the field \( op_{1} \) represents an operator (\text{PlayerFixRunawayProcess} in the example above), and the field \( p_{1} \) represents a precondition (\text{isSelectedNiceTool-42} in the example).

**Prompt Template 1A:** The purpose of the prompt template is to prompt the student to take the action that is required. An NPC or a Text-To-Speech system prompts the student:

‘You need to \text{getDescription(opr1)}.”

For the FixIt example above, \text{getDescription(PlayerFixRunawayProcess)} returns the text: “use the nice tool on the runaway process panel,” which causes the generated text to read: “You need to use the nice tool on the runaway process panel.”

**Hint Template 1A:** This template is designed to provide more detail on why the particular action is required. An NPC or the HUD conveys the hint to the student:

“If \text{getDescription(p1)} then you can \text{getDescription(opr1)}.”

For the FixIt example above, the call to \text{getDescription(isSelectedNiceTool - 42)} returns the text “the nice tool is selected”. The resulting generated text is: “If the nice tool is selected, then you can use the nice tool on the runaway process panel.” Future work could investigate more sophisticated natural language generation techniques to avoid errors in grammar and diction.
Show Template 1A: To avoid cluttering the main flow of description in other parts of this section, the requirements to enable the show template have not been previously described. Specifically, an optional field for each condition in the ZLib allows the Zlib author to specify an action be performed in the ZWorld in the event the “show” remediation is selected for that condition. When a value for this field is specified for a condition for which the show template is selected as the remediation, the action specified in the field is sent to the learning environment for execution. In the case of the PlayerMoveToNiceTool operator in the FixIt learning environment, the show template will cause FixIt to play a cut-scene – a pre-rendered in-game cinematic – showing the location of the runaway process panel.

Teach Template 1A: The teach template explains exactly what the student needs to do and why the student needs to do it. An NPC or the Text-To-Speech announcement tells the student that

“You need to make it so that $getDescription(p1)$ now so that you can $getDescription(opr1)$ because you need to $getDescription(opr1)$ to make further progress.”

For the FixIt example above, the resulting generated text is: “You need to make it so that the nice tool is selected now so that you can use the nice tool on the runaway process panel because you need to use the nice tool on the runaway process panel to make further progress.”

Do Template 1A: The “do” template sends the associated action to the learning environment with an additional indicator that the learning environment itself should
perform the action. In the case of this example in FixIt, the system takes control of the student’s character and uses the nice tool on the process panel.

**Remediation Scenario 1B: Student Inaction - Unknown Effect**

This remediation scenario arises when Annie decides in stage one to attempt to raise the student’s level of belief that a particular operator has a particular effect, where the most recent reduction in that belief was a result of diagnostic scenario 1B, described in section 4.4.1 on page 78. Diagnostic scenario 1B arises when the student fails to take an action that is needed before the rest of the plan can continue.

For example, in the FixIt domain, it may be the case that the student has failed to pickup the “Nice” tool, as the student is unaware that an effect of picking up the “Nice” tool is that he or she will then have the “Nice” tool. This might be reflected in the student model with the following statement:

\[
\langle \langle \text{PlayerMoveToPickupNiceTool}, \text{hasEffect}, \text{hasNiceTool-17} \rangle, \text{HUB} \rangle
\]

For the templates below, the field \( opr_1 \) represents an operator (“PlayerMoveToPickupNiceTool” in the example above), and the field \( e_1 \) represents an effect (“hasNiceTool” in the example).

**Prompt Template 1B:** The purpose of the prompt template is to prompt the student to take the action that is required. An NPC or a Text-To-Speech system directs the student:

“\( \text{You need to \textit{getDescription}(opr1)} \)”)
For the FixIt example above, \( \text{getDescription}(\text{PlayerMoveToPickupNiceTool}) \) returns the text: “move to the location of the pickup for the nice tool”, which causes the generated text to read: “You need to move to the location of the pickup for the nice tool.”

**Hint Template 1B:** The hint template provides more detail on why the particular action is required. An NPC or the Text-To-Speech announcement directs the student:

“If you \( \text{getDescription}(\text{opr}1) \) then \( \text{getDescription}(e1) \).”

For the FixIt example above, the call to \( \text{getDescription}(\text{hasNiceTool}) \) returns the text “you have the nice tool.” The resulting generated text is: “If you move to the location of the pickup for the nice tool then you have the nice tool.” Again, future work may investigate more sophisticated natural language generation techniques to improve the quality of the text.

**Show Template 1B:** As mentioned above, an optional field for each condition in the \( ZLib \) allows the \( Zlib \) author to specify an action to be performed in the event the “show” remediation is selected for a specific condition. When this field is present, and when the show template is selected as the remediation, the action is sent to the learning environment. In the case of the \( \text{PlayerMoveToPickupNiceTool} \) operator in the FixIt learning environment, the show template will cause FixIt to play a cut-scene showing the player performing the pick-up action and holding the nice tool as a result.
**Teach Template 1B:** The teach template explains exactly what the student needs to do and why the student needs to do it. An NPC or the Text-To-Speech announcement tells the student that

“You need to $getDescription(e_1)$ now so that $getDescription(o_{pr1})$ because it needs to be the case that $getDescription(o_{pr1})$ for you to make further progress."

For the FixIt example above, the resulting generated text is: “You need to move to the location of the pickup for the nice tool now so that you have the nice tool because it needs to be the case that you have the nice tool for you to make further progress. ”

**Do Template 1B:** The “do” template sends the associated action to the learning environment with an additional indicator that the learning environment itself should perform the action. In the case of the PlayerMoveToPickupNiceTool operator in FixIt, Annie directs the system to take control of the student’s character and walks it to the pickup location of the nice tool.

**Remediation Scenario 2A: Unknown Precondition**

This remediation scenario arises when Annie decides in stage one to attempt to raise the student’s level of belief that a particular operator has a particular precondition, where the most recent reduction in that belief was a result of diagnostic scenario 2A (described in section 4.4.2 on page 79). Diagnostic scenario 2A arises when the student attempts an action that fails because of an unsatisfied precondition, where that precondition was never falsified, or threatened at any point in the plan history.
For example, in the FixIt domain, it may be the case that the student attempted to select the “Nice” tool (an action that involves placing the “Nice” tool in the hand of the student’s character, allowing him or her to then use the tool) and the action failed, as the student is unaware that a precondition of selecting the “Nice” tool is that the student must first possess the “Nice” tool. This might be reflected in the student model with the following statement:

\[\langle\text{PlayerSelectNiceTool, hasPrecondition, hasNiceTool-18}, \text{HUB}\rangle\]

For the templates below, the field \(opr\) represents an operator (\(\text{PlayerSelectNiceTool}\) in the example above), and the field \(p\) represents a precondition (\(\text{hasNiceTool}\) in the example). Because scenario 2A is the reason for this misconception in the student model, Annie knows that the student attempted action \(opr\) condition and failed because \(p\) was not established.

**Prompt Template 2A:** The purpose of the prompt template is to prompt the student to take the action that is required. An NPC or a Text-To-Speech system could tell the student the precondition that must be satisfied:

“You need it to be the case that \(\text{getDescription}(p)\)”

For the FixIt example above, \(\text{getDescription}(\text{hasNiceTool-18})\) returns the text: “You have the nice tool.”, which causes the generated text to read: “You need it to be the case that you have the nice tool.”

**Hint Template 2A:** This template is designed to provide more detail on why the particular action is required. An NPC or the Text-To-Speech announcement asks the student:

“You cannot \(\text{getDescription}(opr)\) unless \(\text{getDescription}(p)\)”.
For the FixIt example above, the call to \texttt{getDescription(PlayerSelectNiceTool)} returns the text \textquote{use the scroll wheel on the mouse to select the nice tool\textquot; so the resulting generated text is: \textquote{You cannot use the scroll wheel on the mouse to select the nice tool unless you have the nice tool.}''

\textbf{Teach Template 2A:} The teach template for scenario 2A is designed to underscore that \texttt{opr1} requires \texttt{p1} to be true. It is not the focus in this scenario to teach which action(s) could be performed to establish \texttt{p1}. Therefore, the teach template merely emphasizes the importance of \texttt{p1} to \texttt{opr1}. An NPC or the Text-To-Speech announcement asks the student:

\textquote{It must be the case that \texttt{getDescription(p1)} before you \texttt{getDescription opr1).}''

For the FixIt example above, the call to \texttt{getDescription(PlayerSelectNiceTool)} returns the text \textquote{use the scroll wheel on the mouse to select the nice tool\textquot; so the resulting generated text is: \textquote{It must be the case that you have the nice tool before you use the scroll wheel on the mouse to select the nice tool.}''

\textbf{Do Template 2A:} There is no \textquote{do\textquot; template for this misconception. Annie’s job for 2A is not to establish \texttt{p1}, but just to ensure the student knows that \texttt{p1} is a precondition for \texttt{opr1}.

\textbf{Remediation Scenario 3A: Unknown Effect}

This remediation scenario arises when Annie decides in stage one to attempt to raise the student’s level of belief that a particular operator has a particular effect, where the most recent
reduction in that belief was a result of diagnostic scenario 3A (described in section 4.4.3 on page 81). Diagnostic scenario 3A arises when the student’s attempt to execute an action fails because of an unsatisfied precondition, where that precondition was satisfied at some point in the past, but became false due to an intervening action. Scenario 3A specifically focuses on the case where this happens because the student was unaware of the effect of an earlier action that negated the necessary precondition.

For example, it may be the case that the student attempts an action EnterRedGate which fails because the red gate is locked, where an effect of an earlier action unlocking the blue gate causes that the red gate to automatically lock. For the templates below, opr1 represents an operator (UnlockBlueGate in the example above), and the field e1 represents an effect (is-LockedRedGate in the example) and opr2, is operator EnterRedGate. Annie’s job is to remedy the misconception that an effect of opr1 is e1.

**Prompt Template 3A:** The purpose of the prompt template is ensure the student knows the status of e1, which caused the problem. An NPC or a Text-To-Speech system tells the student:

“You cannot getDescription(opr2) if getDescription(e1)”

For the FixIt example above, getDescription(opr2) returns the text “go through the red gate,” and getDescription(isLockedRedGate) returns the text “The red gate is locked.” This causes the generated text to read: “You cannot go through the red gate if the red gate is locked.”

**Hint Template 3A:** This template is designed to provide more detail on why the particular action failed. An NPC or the Text-To-Speech announcement asks the
student:

“If you $getDescription(opr1)$ then $getDescription(e1)$.”

For the example above, the call to $getDescription(UnlockBlueGate)$ returns the text “unlock the blue gate”. The resulting generated text is: “If you unlock the blue gate then the red gate is locked.”

Show Template 3A: As mentioned above, an optional field for each condition in the $ZLib$ allows the $Zlib$ author to specify an action to be performed in the event the “show” remediation is selected for a specific condition. When this field is present, and when the show template is selected as the remediation, the action is sent to the learning environment. In the case of the UnlockBlueGate operator in the FixIt learning environment, the show template will cause FixIt to play a cut-scene showing a result of the UnlockBlueGate is that the red gate becomes locked.

Teach Template 3A: The teach template explains exactly what the student needs to do and why the student needs to do it. An NPC or the Text-To-Speech announcement tells the student that

“If you $getDescription(opr1)$ then $getDescription(e1)$ and you cannot $getDescription(opr2)$.”

For the example above, the resulting generated text is: “If you unlock the blue gate then the red gate is locked and you cannot go through the red gate.”

Do Template 3A: There is no “do” template for this misconception. Annie’s job for 3A is not to establish $\neg e1$, but just to ensure the student knows that $e1$ is an effect of $opr1$. 

69
4.3.3 Stage Three: Action Specification

In stage three, Annie determines which of the actions in the current plan should execute next. Because Annie makes use of a plan representation that models the temporal constraints over steps in a plan as a partial order, there may be more than one potential action that can be executed next (the *proximal actions*), and these actions may be either student or system-initiated.

Because Annie cannot force the student to take a particular action, additional processing must occur when Annie selects a student-initiated action to be executed next. In this case, the Zócalo execution manager processes the directive to execute the action as a null request and does nothing. Because the student has no way of knowing that the system is expecting a particular action, it is not likely that this action will be immediately executed. Consequently, Annie should not progress forward in the tutorial plan until the student is given enough time to choose to execute an action (which may or may not be the one Annie selected).

Therefore, each time a student-initiated action is chosen, Annie starts a timer (currently set to 15 seconds) to give the student time to execute an action. The timer is terminated early if any report of student action arrives. During the wait period however, no other actions are sent by Annie to the execution manager. In this manner, Annie does not spend time exhausting its supply of system-initiated actions or oscillating between alternatives in a way that confuses the student.

When Annie has determined that one or more remedial actions should be executed in the *ZWorld*, the system must also hold off on sending additional actions for execution until the execution manager has reported that the remedial actions have completed their execution. Because the underlying game engine used by FixIt cannot provide specific duration information
for the execution of its actions, Annie also uses a timer-based approach to estimate the length of time to wait for the completion of remedial actions. When remedial actions have been sent to the execution manager for execution, Annie uses a ten second timer to hold off on new directives to the ZWorld. This interval is an upper bound on the execution times of all remediations provided in the current system.

If no remediations were chosen in stage one and stage two and no outstanding timers are running in the system, Annie selects the next action for execution by choosing non-deterministically from the set of proximal actions, giving higher priority to student-initiated actions and breaking ties arbitrarily.

### 4.3.4 Stage Four: Action Execution

If the action chosen in stage three is a system-initiated action, Annie sends a message to the Zócalo execution manager to execute that action. Annie proceeds to stage five while the execution manager asynchronously proceeds to perform the action and report back to Annie when the action completes.

If the action chosen in stage three is a student-initiated action, Annie sends a message to the Zócalo execution manager to wait for the student to execute that action. As described above, the execution manager does nothing to the game state and Annie proceeds directly to stage five.

### 4.3.5 Stage Five: Action Result Processing

In stage five, Annie processes any action results received in its buffers since the last iteration of the execution cycle. If no action results have been received, there is nothing to process in stage five and the execution cycle proceeds directly to its 500 ms wait before returning to
stage one. If any action results have been received, then Annie processes all of them before performing the 500 ms wait and returning to stage one.

The Zócalo execution manager sends a notification message to Annie any time an action succeeds or fails, whether it is initiated by the system or by the student. It also sends a message when an expected student action remains unexecuted after a long wait. Based on the receipt of such a message, Annie updates the execution trace $\mathcal{ET}$ component of the $\textit{ABase}$. When the action being reported is a student action, Annie also updates $\mathcal{SM}$. The student model updates are based on whether the action being reported was successfully executed or not and how the action’s execution context matches a pre-computed catalogue of action result scenarios. Annie’s processing of these action result scenarios is described in this section and each of the scenarios is described in detail in section 4.4, beginning on page 77.

Each action attempted by the student presents an opportunity for Annie to reason about its model of student beliefs $\mathcal{SM}$ about the operators in the $\textit{ZLib}$. Annie bases its reasoning, in part, on the assumption that the student intends for the action to succeed, and believes that the action will succeed. Although it is not difficult to imagine special cases where this assumption would not hold, the net diagnostic value in making this assumption is positive.

If a reported student action has failed, Annie assumes that the student has at least one misconception in their mental model about the action’s execution context. It may also be the case, however, that this action attempt presents evidence in support of one or more \textit{correct} beliefs. For example, an action which fails due to a single un-established precondition may nevertheless provide positive evidence for parts of the student model that are associated with other, fulfilled preconditions of the same action. Likewise, successful actions may attest to both correct and incorrect beliefs in the student’s mental model. When student inaction is detected, it indicates
that the student is either unaware of the significance of the action, or has a misconception about the preconditions or effects of other actions. In any of these cases, Annie first narrows the action result down to the most appropriate scenario and then updates the beliefs in the $SM$ based on the conditions matched in the scenario description.

The following seven scenarios provide an enumeration of the execution contexts where Annie can update its student model based on student activity (or inactivity).

**Scenario 1: Student Inaction** When a player-initiated action is selected for execution, the system waits for a pre-defined period of time (in the learning environment for the current implementation, this wait is 15 seconds) for that action to be taken by the student. If the student takes any action during the wait period, the Zócalo execution manager sends a message to Annie describing the action result and the wait on the chosen action is terminated. If the wait expires without the student taking the chosen action, the Zócalo execution manager sends a message to Annie that the action failed due to a timeout condition.

**Scenario 2: Failed Action - Unthreatened Precondition** This scenario arises when an action that the student attempt fails, because at least one precondition for the action was never established, and that failed precondition has been false since the initial state (an unthreatened precondition).

**Scenario 3: Failed Action - Unresolved Threat** This scenario arises when an action the student attempts fails because one of its preconditions was established by an earlier action but then undone by some intervening action.

**Scenario 4: Successful, but fatally flawed action** In this scenario, the action executed by the student succeeds, but Annie determines that its execution eliminates all possible success-
ful plans from the plan space (e.g., the student just destroyed an irreplaceable resource required for successful completion of the plan). The student may not realize that all success paths have been eliminated by this action, so successful remediation may be more difficult.

**Scenario 5: Successful, but unnecessary action** A successful student action may reveal a misconception if its execution was not a necessary step toward the goal. If Annie determines that an action is not a necessary constituent of the successful set of plans \( PPSS \), then it may be the case that the student has a misconception about the effects of an earlier action.

**Scenario 6: Successful, but suboptimal action** A successful action sequence can also reveal a misconception in the case that a more optimal sequence of actions can achieve the same goal. For example, if the student performs a chain of actions that bring about the same result that a single action could achieve, it could be a result of the student not knowing the effects of that action.

**Scenario 0: Complete Success** This is the normal success case, meaning the student successfully performed an action without negative side-effects.

**Student Model: Updating Beliefs**

This section describes generalized scenarios of action success and failure. Each time a student-initiated action succeeds or fails, Annie considers all of these scenarios in determining which belief likelihood estimates should be updated. A given scenario can suggest changes to the belief likelihood estimates of more than one \( ZBel \). Within each scenario description, each specific belief change is accompanied by a unique diagnostic template identifier. Although Annie considers all of the scenarios in analyzing a given action result, a particular \( ZBel \) in \( SM \),
will match only a single diagnostic template.

The suggested changes in each diagnostic template take the form of either decreases or increases to a particular belief likelihood estimate. Where the scenario description indicates a probably student misconception, the template will suggest decreasing the belief likelihood estimate for the \( ZBel \) in the \( SM \) corresponding to that \( ZFact \). Conversely, where a scenario indicates support for the student knowing a particular \( ZFact \) a diagnostic template will suggest increasing the belief likelihood estimate for the \( ZBel \) in the \( SM \) corresponding to that \( ZFact \).

A simplistic calculus is used to enact these increases and decreases in belief likelihood estimates. The belief strength model used in the current implementation of Annie and described in section 4.1 and shown in table 4.1 on page 46 is a set of discrete categories. In the current implementation, an increase or decrease in belief likelihood suggested by the diagnostic scenarios results in an increment or decrement of exactly one belief likelihood level. An exception is that an increase will not raise a belief likelihood above its maximum possible level nor will a decrease lower a belief likelihood below its minimal possible level.

Annie adjusts the student model representation of these belief levels as shown in the following example. If the \( SM \) currently contains the \( ZBel \):

\[
\langle \langle \text{PlayerMoveToPickupNiceTool, hasPrecondition, isToolSpawnLoc–14}, \text{NB} \rangle \rangle
\]

but a misconception is identified that indicates the student may be unaware of the precondition, and it calls for a decrement of the belief likelihood estimate, the corresponding \( ZBel \) would be changed to:

\[
\langle \langle \text{PlayerMoveToPickupNiceTool, hasPrecondition, isToolSpawnLoc–14}, \text{UB} \rangle \rangle
\]
Conversely, if the $SM$ currently contains the $ZBel$:

$\langle \langle \text{PlayerSelectNiceTool, hasPrecondition, hasNiceTool-18} \rangle, \text{NB} \rangle$

and the player successfully performs the “PlayerSelectNiceToolAction” such that the remediation scenario suggests that belief likelihood estimate in the corresponding $ZBel$ should be increased, Annie would change this tuple to:

$\langle \langle \text{PlayerSelectNiceTool, hasPrecondition, hasNiceTool-18} \rangle, \text{LB} \rangle$

As Annie is extended by future work to employ to an alternative model of belief, (e.g. a Bayesian Belief Network [18]), a more sensitive calculus of belief changes will be better supported in the diagnostic templates. In fact, Bayesian Belief Networks (BBNs) have been used in exploratory ITS in the past to model the uncertainty of the student model. Bunt and Conati [10] employ BBNs for a student model in the domain of math, and Mott [49] uses a Dynamic Decision Network (DDN) to model student knowledge in the domain of microbiology. Albrecht and Zukerman [1] use BBNs for student plan recognition in an adventure game. Clearly, there is precedent for a more fine-grained model of student belief than that chosen in the current implementation.

The student model maintains a full history of belief changes for each element of the student beliefs. When the belief is changed as a result of one of these remediation templates, the identifier of the misconception (e.g., “2A” is the identifier for the “Unknown Precondition” misconception) is recorded along with the change. This allows a more precise remediation to be chosen if the misconception is later selected for remediation.
4.4 Diagnostic Scenarios

This section describes a collection of schemas that characterize the task execution contexts where Annie can update its student model. These schema correspond to the seven scenarios listed earlier in section 4.3.5 on page 73. Each time a student-initiated action succeeds or fails or is unexecuted after a long wait, Annie considers all of these scenarios in determining which beliefs should be updated. However, for each update of a particular ZBel in SM, only a single diagnostic template, particular to a single diagnostic scenario will be selected.

Each scenario contains a text description, a figure depicting the planning perspective of the scenario, and a table showing the particular diagnostic template identifiers to be associated with the change for possible processing of remediations s and the suggested belief likelihood estimate changes (using the abbreviations “incr” for increment, and “decr” for decrement).

This catalogue is not comprehensive in that the student may also have misconceptions about “phantom” actions having executed, or “phantom” effects or conditions of operators, or misunderstandings regarding the order in which a particular sequence of actions occurred. Future work will investigate providing a method for reading in diagnostic templates from external files, and at that point consideration will be given to providing sufficient expressive power to make the library more comprehensive.

The following sections describe, for each of the diagnostic scenarios, the schema and templates that can be used to update Annie’s student model.
4.4.1 Scenario 1: Student Inaction

When a player-initiated action is selected for execution, the intention is that the system wait for some period of time (in the learning environment for the current implementation, this wait is 15 seconds) for that action to be taken by the student. If the student takes any action during the wait period, the Zócalo execution manager sends a message to Annie as usual describing the action result and the wait on the chosen action is terminated. If the wait expires without the student taking the chosen action, the Zócalo execution manager sends a message to Annie that the action failed due to a timeout condition. Given that the planner had nominated this action as a proximal step on the path toward plan success, and the student did not perform the action, there is a strong probability that the student has a misconception. Either the student is not aware that the effects of previous actions established the preconditions for this action, the student does not realize this action is required because it establishes effects required for the rest of the plan.

Figure 4.2 shows a situation where a student-initiated action $B$ has not yet been performed, but is needed before the rest of the plan can continue. This would arise when the learning environment detects a timeout after waiting for a student-initiated action. It is likely that the student is unaware of one or more of the preconditions and effects of this action. To trigger appropriate remediation, belief levels are decremented for all the preconditions and effects of this operator.
Table 4.2: Student Inaction

<table>
<thead>
<tr>
<th>Template Identifier</th>
<th>Nickname</th>
<th>Belief Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>Unknown Precondition</td>
<td>decr(B, hasPrecondition, ( p_0 ))</td>
</tr>
</tbody>
</table>
| 1B                  | Unknown Effect    | decr(B, hasEffect, \( p_1 \))  
                          decr(B, hasEffect, \( p_2 \)) |

Figure 4.2: Student Inaction

4.4.2 Scenario 2: Failed Action - Unthreatened Precondition

When an action that the student attempts fails, there is a good probability that an underlying misconception can be identified, because the system can identify at least one precondition that was not established. Actually, there are two classes of failure: one in which the precondition has been false since the initial state (an unthreatened precondition), and one in which the precondition once held but was subsequently falsified. This scenario covers the former case.

Figure 4.3 depicts the initial state conditions and an execution trace that lead up to a student-initiated action attempt \( D \) that fails due to unsatisfied precondition \( r \), where the precondition \( r \) was never falsified or threatened at any point in the plan history. Note that in each of these
failure cases, we have a positive update to make to the student model regarding proposition \( p \) as a precondition of \( D \) and an effect of \( C \) (assuming both are student-initiated actions).

<table>
<thead>
<tr>
<th>Template Identifier</th>
<th>Nickname</th>
<th>Belief Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>2A</td>
<td>Unknown Precondition</td>
<td>( \text{decr}(D, \text{hasPrecondition}, r) )</td>
</tr>
<tr>
<td>2K</td>
<td>Known Precondition</td>
<td>( \text{incr}(D, \text{hasPrecondition}, p) )</td>
</tr>
<tr>
<td>2L</td>
<td>Known Effect</td>
<td>( \text{incr}(C, \text{hasEffect}, p) )</td>
</tr>
</tbody>
</table>

Figure 4.3: Unthreatened Precondition
4.4.3 Scenario 3: Failed Action - Unresolved Threat

Figure 4.4 shows an action attempt C that fails due to a precondition $p$ that was true at some point but later became falsified or threatened. The belief updates suggested in the case of an unresolved threat are depicted in table 4.4. Action failures for a given precondition will match either this scenario (3), or the previous, unthreatened precondition scenario (2), although a given action might one or more preconditions in each category.

Table 4.4: Action Failure - Unresolved Threat

<table>
<thead>
<tr>
<th>Template Identifier</th>
<th>Nickname</th>
<th>Belief Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>3A</td>
<td>Unknown Effect</td>
<td>$\text{decr}(\text{B, hasEffect, } \neg p)$</td>
</tr>
<tr>
<td>3B</td>
<td>Unknown Precondition</td>
<td>$\text{decr}(\text{C, hasPrecondition, } p)$</td>
</tr>
</tbody>
</table>

Figure 4.4: Unresolved Threat
4.4.4 Scenario 4: Successful, but *fatally flawed* action

In some cases, a student’s action may succeed, but Annie may detect that its execution will eliminate all possible successful plans from the plan space (e.g., the student just destroyed an irreplaceable resource required for successful completion of the plan). The student may not realize that all success paths have been eliminated by this action, so remediation may be more difficult.

Figure 4.5 depicts an action attempt $E$ that succeeds, but introduces a flaw ($\neg p$) that cannot be resolved and thus dooms all possible successful plans. For the purposes of this illustration, we assume that the only path to establish condition $r$ is through operator $F$, which has precondition $p$, and that there is no other way to establish $p$ once it has been falsified by $E$. The twin vertical lines represent a block to further progress. Table 4.5 depicts the corresponding belief updates.

<table>
<thead>
<tr>
<th>Template Identifier</th>
<th>Nickname</th>
<th>Belief Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>4A</td>
<td>Unknown Goal</td>
<td>$\text{decr}(\text{Goal, hasPrecondition, } r)$</td>
</tr>
<tr>
<td>4B</td>
<td>Unknown Precondition</td>
<td>$\text{decr}(F, \text{hasPrecondition, } p)$</td>
</tr>
<tr>
<td>4C</td>
<td>Unknown Effect</td>
<td>$\text{decr}(E, \text{hasEffect, } \neg p)$</td>
</tr>
</tbody>
</table>
4.4.5 Scenario 5: Successful, but unnecessary action

Successful actions can reveal misconceptions when Annie consults the planning problem solution set \( \mathcal{PPSS} \). For even though the executed action succeeds, if it is not relevant to achieving the goal it may show that the student is unaware of some of the conditions that have already been established.

For example, in figure 4.6 attempt C succeeds, but was not necessary, as \( q \) had already been established. Three belief updates are indicated as shown in table 4.6.

<table>
<thead>
<tr>
<th>Template Identifier</th>
<th>Nickname</th>
<th>Belief Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>5A</td>
<td>Unknown Effect</td>
<td>( \text{decr}(D, \text{hasEffect}, q) )</td>
</tr>
<tr>
<td>5K</td>
<td>Known Effect</td>
<td>( \text{incr}(C, \text{hasEffect}, q) )</td>
</tr>
<tr>
<td>5K</td>
<td>Known Precondition</td>
<td>( \text{incr}(E, \text{hasPrecondition}, q) )</td>
</tr>
</tbody>
</table>
4.4.6 Scenario 6: Successful, but suboptimal action

It is possible for a student to execute a series of successful actions, where none of the individual actions are problematic, but the length of the chain may show that the student was unaware of a much easier way to reach the same goal. This is a difficult situation to properly diagnose. Because an analysis of the the ZLib used in the evaluation environment showed that it could not give rise to this type of scenario, this template was not implemented. Nevertheless, there is a useful student model update that could be gleaned, as shown in figure 4.7 and table 4.8.
Table 4.7: Action Success - Suboptimal Plan

<table>
<thead>
<tr>
<th>Template Identifier</th>
<th>Nickname</th>
<th>Belief Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>6A</td>
<td>Unknown Effect</td>
<td>$\text{decr}(A, \text{hasEffect}, p_9)$</td>
</tr>
<tr>
<td>6K</td>
<td>Known Precondition</td>
<td>$\text{incr}(E, \text{hasPrecondition}, p_9)$</td>
</tr>
</tbody>
</table>

Figure 4.7: Suboptimal Plan

4.4.7 Scenario 0: Complete Success

When a student successfully executes an action and none of the qualified success scenarios (numbers 4, 5, or 5) apply, then the action represents an unqualified success. In this case the
belief likelihoods associated with all the preconditions of the actions as well as the effects of previous actions that satisfied these conditions are incremented. For the successful execution of action F in figure 4.8, the corresponding belief likelihood changes are depicted in table 4.8.

Table 4.8: Action Success - Correct Beliefs

<table>
<thead>
<tr>
<th>Template Identifier</th>
<th>Nickname</th>
<th>Belief Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A</td>
<td>Known Precondition</td>
<td>incr(F, hasPrecondition, p)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incr(F, hasPrecondition, q)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incr(F, hasPrecondition, s)</td>
</tr>
<tr>
<td>0B</td>
<td>Known Effect</td>
<td>incr(C, hasEffect, p)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incr(G, hasEffect, q)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>incr(B, hasEffect, s)</td>
</tr>
</tbody>
</table>

Figure 4.8: Diagnosing Correct Beliefs
4.5 Design Summary

A common theme emerges in the preceding discussion of the detailed design. Annie’s current implementation focuses on the value of planning knowledge in guiding the tutorial. Several opportunities for increasing the intelligence of Annie’s reasoning were bypassed in this implementation, yet Annie still performed well in the evaluation (as will be discussed in Chapter 5). This suggests that the plan reasoning at the core of Annie provides a solid and effective foundation for making decisions about the timing and nature of scaffolding in task-focused environments. Future work can build on this foundation to evolve Annie’s intelligence and evaluate the effectiveness of alternative design enhancements.

For future work, implementers may consider a more expressive model of probabilistic belief, more complex scenarios for diagnosing student misconceptions, and more powerful forms of remediation that harness more of the learning mechanisms typical of digital games. Reasoning about variable bindings in operators would likely add significant expressive power to both diagnosis and remediation, benefits that would need to be weighed against increased performance costs and knowledge engineering requirements.
Chapter 5

Evaluation

Two separate formal evaluations have been made of Annie. The first evaluation validated that Annie can accurately perceive student misconceptions about a domain solely through passive observations of student behavior. A deliberately simplified version of the primary evaluation domain was used to help focus the study on diagnostic capabilities. Data collected from sixteen human subjects in this pilot evaluation showed that Annie’s diagnostic accuracy was equivalent to that of a trained human observer.

A second formal evaluation was held two months after the first evaluation to assess Annie’s impact on student learning effectiveness. In this evaluation, the FixIt learning environment was extended to a full game consisting of a sequence of three learning “challenges,” or missions, as described in Appendix A, and the Annie system was engaged in both diagnosis and remediation of student misconceptions. Additional updates were made to FixIt as a result of the pilot study specifically to help students better understand the visual metaphors and intended semantics of the learning environment.
5.1 Evaluation Domain

Because Annie specializes in task-based learning, it is important for it to be evaluated in a domain that contains multiple causally-related actions. An exploratory learning environment was created for the domain of computer security as viewed from the perspective of a generic operating system. Built on the Unreal Tournament 3 engine, this immersive 3-D learning environment, called ‘FixIt,’ animates the message traffic between running processes and various components of a fictional computer. The computer is depicted as a green circuit board with brightly reflective, platinum colored pathways connecting its various external components to a central processing hub, as shown in an “aerial” view in figure 5.1.

![FixIt: Top Down View](image)

Figure 5.1: FixIt: Top Down View
5.1.1 Essential Actors in the Domain

The processes in the learning environment are depicted as translucent colored panels arranged in an outward facing circle from the central hub of the CPU. Each process has a distinct name, graphical icon, and color. As processes send and receive messages to various parts of the computer, the packets comprising these messages are depicted as box-shaped mouse droids matching the color and iconography of the corresponding process that emerge from underneath the corresponding panel (if outgoing from the CPU) or at the corresponding system component (if incoming toward the CPU). Mouse droids slide along the platinum-colored circuit pathways to provide visual indications of the level and direction of messaging traffic generated by each process. Processes only dispatch mouse droids when they are become active, where process activation is depicted by a brightly animated shield effect light that rotates sequentially around the hub to symbolize the round-robin activation of each process. Figure A.2b shows a mouse droid being dispatched from a process. Process panel locations, droid paths, and messaging frequency are all programmatically defined, so that these parameters can manipulated or even randomized at run-time according to the specifications of a given learning exercise.

5.1.2 Potential Student Interactions with the Domain

Students have three primary ways to interact with the learning environment. First, they can observe the environment to understand the details of what is happening in a given learning exercise, and generate ideas for actions to be taken. Second, they can move through the environment to gain better perspective or put themselves in position to take particular actions. Occasionally, movement can result in a state change in the world, when the player’s avatar moves over a “pick-up,” a particular location where the character implicitly gains possession of an object at that location. These first two modes of interaction are provided through as is
typical in many computer games, through simultaneous control of keyboard keys for character movement, and the mouse for moving the “camera”, or first person view.

The third method for the student to interact with the learning environment is through a set of “tools,” which are basically re-purposed and re-labeled weapons from the Unreal Tournament arsenal. For example, the Information Tool, when fired on a process panel, causes a translucent pop-up window to be rendered on the student’s Heads Up Display, or HUD, that provides a text description of some of the operational details of that process. Similarly, when fired on a mouse droid, the Information Tool shows text describing the source and destination locations of the mouse droid, its owning process, its payload type and a sample of its payload data. Although the process panel and mouse droid behaviors were common to both evaluations, seven different
types of tools have been developed for “FixIt,” and the two evaluations differed in the set of tools made available to the student. Therefore, descriptions of these tools are deferred until the corresponding sections.

5.2 Pilot Evaluation: Diagnostic Capabilities

5.2.1 Synopsis

The first formal evaluation of Annie assessed the accuracy with which Annie is able to estimate human subjects’ understanding of the cause and effect relationships in an exploratory learning environment. Annie initiated the tutorial session, and updated its student model based on student actions, but was restricted from providing any assistance to the student. Upon completion of the test, the student’s knowledge was evaluated in three ways. First, the student completed a written evaluation to assess their understanding of the of twenty four specific facts about the game environment. Second, Annie provided a full listing of its model of student knowledge at completion of the test. Third, a human expert was shown a full video capture of each student’s test session and given the same paper evaluation as presented to the student. The human observer’s assessment was then compared to the student self-assessment to appraise the accuracy of the human observer. Similarly, Annie’s assessment was compared to the student self-assessment to appraise its accuracy. Annie’s overall accuracy found to be equivalent to that of the human expert at a high level of statistical significance, and a strong correlation was found between Annie’s subject-by-subject accuracy and the accuracy of the human expert.
5.2.2 Pilot Evaluation: Experimental Design

Sixteen adult college graduates volunteered to participate in the study. The subjects had varying degrees of familiarity with computer science, digital games in general, and first-person shooter games in particular. Each evaluation session involved an individual participant interacting with the system in a private room under the supervision of a single experimenter. A two page description of the game controls and mission objectives was presented to the subject at the beginning of the session and subjects were given as much time as necessary to read the instructions before the evaluation began. The instructions remained available to the student for the duration of the evaluation, placed on the desk directly adjacent to the computer. Page 1 of the instructions is shown in figure 5.3 of page 94 and page 2 of the instructions is shown in figure 5.2.2 on page 95.

5.3 Subject Knowledge Self-Assessment

Also described in section 5.2.2, upon completion of the game, each student was asked to fill out an assessment of their knowledge of the game environment. The first page of this self-assessment is shown in figure 5.3. The experimenter provided no additional information to the subject. The subject was asked to read this set of instructions and was given as much time to study it as they liked. When the subject indicated a readiness to continue, the game was started. Unknown to the subject, the game automatically terminated after five minutes, at which point the subject was asked to complete a written questionnaire that assessed their knowledge level of actions, preconditions, and effects central to the game.

For the purposes of the pilot test, Annie was blocked from providing any help to the student, but continued to update its model of student knowledge through passive observation of student
THE STORY BEHIND FixIt

Thank you for volunteering for this pilot test. The story behind this game is that it takes place dozens of years in the future and you and the other volunteers have been drafted into service because we are all on a space station together, and our computer system security has been compromised. Not only have we had to disable our normal automated security system, but also, we can no longer rely on our internal security personnel as several have turned out to be traitors and we do not know whom to trust.

Thus, we have asked you volunteers to use a primitive, early century, human-driven backup system to try to keep our space station computer system running. On this system, you move an avatar through a 3-D representation of what is going on in our real computer. You move your character with the 4 directional arrow keys (or W-A-S-D if you know those) and rotate your view by moving the mouse side to side.

Inside the system are several “tools”, which look a lot like guns floating and turning a few feet off the ground. You can “pick up” these guns simply by walking into them. The tools will automatically be assigned numbers and you can switch from using one to another by typing the number of the tool on your keyboard or using the scroll wheel of the mouse.

Unfortunately we are not exactly sure what these “tools” do. We have heard that there are two different modes of operation for each tool, invoked with the left or right mouse buttons.

The important part of the system is a circular area in its center, with large translucent screens that represent the various programs running on our system. The work being done by these programs is represented by droids produced by each program that then move out to various parts of the system.
You should try to use these tools on the program screens and the droids to learn what they do.

Here is the catch: you will only be able to use this backup system for a few minutes before the connection is broken, because the hardware is so old. When that happens, you will see a red corona form on your view screen and then it will disappear.

If we detect any anomalies while you are hooked up to the system, one of our operatives will use a back door audio channel to warn you. You may be asked to pause, destroy, speed up or speed down a program. Good luck!

When it starts you will be next to an elevator pad that will lowers you into the main section if you walk on to it. From there, try to find some tools and see what they do to brightly colored objects.

CONTROLS SUMMARY

- Arrow keys (or W-A-S-D) move your character. Holding a key down will make it walk in that direction. **Hint:** Use the mouse with your right hand, and keep three fingers of your left hand on the movement keys the whole time:
  - Ring finger on A (or left arrow)
  - Middle finger on W (or up arrow) to move forward, occasionally on S (down) to go back
  - Index finger on D (or right arrow)
- Moving the mouse side to side rotates your view
- Mouse left and right buttons **often do different things** when you use them to operate (fire) each tool (gun)
- If you get your character stuck and are unable to move, try hitting space bar to jump while moving.
actions in the learning environment. To compensate for the increased difficulty posed by the lack of help, the FixIt learning environment was simplified by adding a wall to block the user from exploring the disc drive area, which reduced the explorable area of the game by fifty percent. The game plan was also simplified, so that the student was asked simply to slow down one runaway process: a variant of the first learning exercise “mission” of the full game.

Close to where the student “spawned,” or began the game were pick-up locations for five different types of “tools.” As shown above, the game instructions made a strong suggestion that the student should acquire and test the effects of these tools on the panels and droids. In fact, knowledge of the different effects of these five tools on the panels and droids comprised the majority of the student knowledge assessment.

5.3.1 Pilot Evaluation: Results

As each session ended, Annie output the entire contents of its student model to a log file. The experimenter translated specific elements of this student model to corresponding questions on the student’s self-assessment. For example, question #8 asked the student to specify what happened when the left mouse button was used to apply tool number 3 to a particular object in the game. The question offers six choices to the student including the correct answer: “slowed it down.” Annie’s student model contained an entry for the likelihood that the student is aware that an effect of the tool 3 action is to slow objects down. If this likelihood was positive, it was interpreted as Annie predicting that the student would correctly answer question #8.

In addition, each game session was recorded with FRAPS video capture software. A volunteer unfamiliar with Annie’s design, but well versed in game play and game design was trained for several hours in the use of the game, including all the potential actions and learning objec-
tives accessible to the subjects of the study. Upon completion of this training, this volunteer was deemed a human expert in the functionality of the game. This human expert watched and listened to the full FRAPS session recording of each subject. Upon completion of each viewing, the human expert completed the same questionnaire given to the subject, with a goal of emulating the subject’s responses, based on the expert’s assessment of the student’s actions with the game.

Thus, there were three versions of the same questionnaire, one filled out by the subject, one by a human expert, and a third through a transliteration of Annie’s student model. The experimenter then compared how accurate the human expert and Annie were in predicting how each student filled out their questionnaire. The results are depicted in table 5.1. For each subject, the first column shows the percentage of the student’s answers to the twenty four questions were accurately predicted by Annie, and the second column shows the same for the human expert. Because there were only 24 questions, the table contains several repeated values. For example, 91.67% accuracy occurs when all but two of the answers were correctly predicted.

Averaging over all subjects, Annie correctly predicted an average of 76% of student responses which compared to an average accuracy of 75% for the human observer. Furthermore, the student-by-student correlation between Annie and the human expert was 0.89 at a significance level of $p < 0.0001$. Annie’s assessments were highly consistent with those of the human expert at a statistically significant level. In a few instances Annie outperformed the human expert when the human failed to remember all of a subject’s actions. Conversely, the human occasionally noticed character actions or visual occlusions unobserved by Annie. The overall correlation of 0.89, however, showed that the student model Annie builds and uses during game
execution can be as accurate as that supplied by a human observer.

<table>
<thead>
<tr>
<th>Subject Code</th>
<th>Annie’s Student Model</th>
<th>Expert Human Observer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject 1</td>
<td>91.67%</td>
<td>91.67%</td>
</tr>
<tr>
<td>Subject 2</td>
<td>75.00%</td>
<td>70.83%</td>
</tr>
<tr>
<td>Subject 3</td>
<td>87.50%</td>
<td>83.33%</td>
</tr>
<tr>
<td>Subject 4</td>
<td>87.50%</td>
<td>91.67%</td>
</tr>
<tr>
<td>Subject 5</td>
<td>79.17%</td>
<td>91.67%</td>
</tr>
<tr>
<td>Subject 6</td>
<td>95.83%</td>
<td>91.67%</td>
</tr>
<tr>
<td>Subject 7</td>
<td>70.83%</td>
<td>75.00%</td>
</tr>
<tr>
<td>Subject 8</td>
<td>75.00%</td>
<td>79.17%</td>
</tr>
<tr>
<td>Subject 9</td>
<td>66.67%</td>
<td>54.17%</td>
</tr>
<tr>
<td>Subject 10</td>
<td>66.67%</td>
<td>62.50%</td>
</tr>
<tr>
<td>Subject 11</td>
<td>58.33%</td>
<td>58.33%</td>
</tr>
<tr>
<td>Subject 12</td>
<td>70.83%</td>
<td>66.67%</td>
</tr>
<tr>
<td>Subject 13</td>
<td>70.83%</td>
<td>70.83%</td>
</tr>
<tr>
<td>Subject 14</td>
<td>95.83%</td>
<td>91.67%</td>
</tr>
<tr>
<td>Subject 15</td>
<td>41.67%</td>
<td>50.00%</td>
</tr>
<tr>
<td>Subject 16</td>
<td>83.33%</td>
<td>70.83%</td>
</tr>
</tbody>
</table>

5.3.2 Pilot Evaluation: Conclusions

Several factors contributed to Annie’s strong performance. First, care was taken to ensure that the learning tasks were not so easy that a majority of students would be fully successful in achieving all the learning objectives. It would be unsurprising for Annie to assess the student’s knowledge as substantially accurate in these cases, producing a ceiling effect on results. Moreover, Annie is not designed to provide help for items perceived to be known by the student, but is tuned to act when there is significant uncertainty about the state of the student’s knowledge.
Therefore, the pilot study was intended to allow only enough time for students to learn an average 25-50% of the game content. The reason for such a low target is that it this is exactly the crucial range of knowledge incompleteness most relevant to Annie’s prospective deliberations.

A second reason the game was set to have a rather short duration was so that the subjects’ self-assessments of in-game knowledge were still accessible in short-term memory. Unfortunately, it proved more difficult to complete substantial portions of the game in the time allotted than anticipated, as our subjects learned an average of only 22% of the entire content. This relative paucity of understanding provided a diagnostic advantage to Annie and the human expert in that there was a higher probability in correctly predicting that the student was unknowledgeable about actions that they never performed.

A notable limitation of the evaluation is that only a single human expert was employed. Therefore, there is insufficient data to attest to how the assessments of this human expert compare to other potential human experts. If this human expert used in this study was exceptionally incompetent, it would make Annie’s performance seem stronger than it should, and vice-versa. An interesting idea for future exploration is to compare how the variance between a set of humans compares to the variance between Annie and a single expert observed in this study. Another avenue for future investigation is to compare the diagnostic accuracy of humans who have domain expertise but are new to the game, to those without prior domain knowledge who are familiar with the game. This could help prioritize the types of work that could improve Annie’s diagnostic accuracy.

An unexpected impact of the pilot evaluation was to influence the design of game elements constituent to the main evaluation. Informal discussions with students following pilot evaluation sessions revealed that consistent and unintended challenges associated with student’s
grasping the semantics of the abstractions and symbols central to the game. Specifically, what was being represented by the process panels, the mouse droids, and how these related to each other was not always well understood. It was never intended for Annie to teach the mapping between game elements and what they symbolized. Rather, Annie is tasked with reasoning about how things happen within the learning environment. As described in 5.4, visual and behavioral aspects of the game were changed to emphasize the core concepts of the game, and a narrated tutorial video was added to ensure that student’s had a better understanding of the game world prior to each evaluation session.

In summary, the pilot evaluation found a strong and significant correlation between Annie and a human observer in diagnosing students’ knowledge acquisition in an exploratory learning environment.
5.4 Main Evaluation: Scaffolding Effectiveness

The main formal evaluation to assess Annie’s impact on student learning effectiveness was held two months after the pilot evaluation of its diagnostic capability. This evaluation used a more complex version of the FixIt game than that used for the pilot evaluation. In addition, the experimental methodology differed from that used in the pilot.

The role of the student in the version of FixIt used in the main evaluation is to find and repair problems in the fictional computer by using the appropriate tools on the appropriate objects and the appropriate times. A sequence of three learning challenges, or missions, guide the student through the learning content. The challenges build on each other so that the student must reuse skills acquired in earlier missions. In general, Annie can benefit from learning challenges that build on earlier challenges, as it extends the relevance of the student model, providing more opportunities for scaffolding and fading than would be possible if the tasks were more discrete.

The first learning challenge involves a runaway process, which can be fixed by the student applying the “Nice” tool to the affected process panel. The second scenario features a SPAM-bot, a malicious program that sends its droids to the system mail program, which obligingly dispatches its own droids carrying the SPAM payload out to the network. The student can solve this problem by using the Delete tool on the SPAM-bot process. Incorrectly using the Delete tool on the mail process will not solve the problem, as the SPAM-bot will continue to send droids to the network disguised as mail droids.

In the third scenario, a different malicious process is created that creates a child process that floods the network with illicit traffic. If the student uses the Delete tool on the infected process, a warning message declares that a system restart is required, lights flash and an alarm sounds.
If the student invokes a restart at that point, the parent process will re-spawn the malicious child when the restart completes. To remove the parent process the student must first delete its startup file from the disc drive, but this file cannot be deleted until after the child process is deleted, which means the student must ignore the lights and alarms until after the file is deleted. A complete description of the game is provided in Appendix A.

5.4.1 Main Evaluation: Experimental Design

As in the pilot evaluation, each session involved an individual participant interacting with the system in a private room under the supervision of a single experimenter. The population of volunteers, however, was more homogenous than that of the pilot study. In the main evaluation, volunteers were drawn from current undergraduates and graduate students in the Computer Science department at North Carolina State University currently enrolled in classes concerning digital game design. Experimental participation constituted a portion of the final grade in these classes, although an alternate activity was also provided so that none of the experimental subjects felt coerced. This set of subjects was familiar with basic concepts of computer science, and in the operation of digital games.

Each evaluation session progressed through the following sequence of phases:

**Phase I**  Pre-assessment: Students were welcomed, and the IRB consent form was reviewed. Students were advised that they could stop the study at any time and still receive the experimental participation credit for their course, if applicable. To preserve anonymity, coded identification numbers were assigned to the subjects and these codes were entered on all written evaluation materials, as well as being manually entered into Annie to be included on all game logs. The subject was then asked to complete a one page written
pre-assessment, included in Appendix C, to gauge pre-test familiarity with the domain of operating systems concepts and computer malware.

**Phase II** Video Orientation: Upon completion of the pre-assessment, a two minute narrated video walk through of the game world and its core concepts was shown to each participant. The video served several purposes. First, it allowed all necessary orientation instructions to be given to each participant in a completely consistent manner. Second, the narration provided a natural conduit to associate the visual representations used in the system with their intended counterparts in the domain of operating system and malware. Third, the video was judged to be a very time-efficient way to orient the subjects to the visual representations used in the game, as it allowed jump-cutting to the specific points in the game where additional explanation was required.

The orientation video explained that the game was a representation of what happens inside a computer. It identified the colored panels as processes, which it defined to be instances of running programs. As the white glow effect rotated into view, the narrator mentioned that this signified each process’s turn to do work. As a mouse droid was launched from a process, the narration described the relationship between processes and the droid. As the avatar’s camera was panned to show the possible droid destinations, the narrator cautioned that those destinations are not reachable by the player. This advice was motivated by a finding in the pilot study, where several subjects felt that the pathways were indicators to move in those directions and thus wasted a lot of time trying to figure out why they were being led away.

The video narrator emphasized that the job of the subject was to monitor the system’s health and diagnose any problems through observation of the droids and the process pan-
els. Then the narrator introduced the scoring system. Full system health is 100. As problems occur this number decreases. The subject receives five points every few seconds when the health is at 100, and fewer points as this decreases. The video explained that the system health score will decrease if the player harms healthy portions of the system. The scoring mechanism was added to discourage a strategy seen with a few subjects in the pilot project who fired indiscriminately on all processes and droids, and to help focus subjects’ attention on the goals of the learning challenges, as the system health would be decreased by 25 or 50 points each time a challenge started.

**Phase III** Experimental Treatments: Each subject was randomly assigned to one of three treatment groups. As described below, each treatment group differed only in the nature of the help-specific interaction provided by the system. Neither the subject nor the experimenter received explicit indications of the chosen treatment. The three treatment groups were:

**Full Annie:** Annie was allowed to provide help whenever and however it felt necessary. For this study, any chosen remediation (which usually took the form of a text prompt combined with text-to-speech narration of the same prompt) was repeated once, after a ten second delay in the Full Annie treatment. In the Ablated Annie condition, the prompt was only issued once. The one other difference in the Full Annie treatment was that for the actions involving finding tools, after two prompts a cut-scene was displayed showing the location of the tools.

**Ablated Annie:** Annie was modified to provide only one remediation per observed misconception, as opposed to the two or three remediations that could occur for a single misconception in the Full Annie version.

**Control (no Annie-provided assistance):** Subjects progressed through the FixIt game,
with no assistance beyond the Phase II instructions common to all three treatments.

The initial plan was to apply the experimental treatment of Phase III to cover the first three learning missions in FixIt, where a fourth learning mission would then be used for Phase IV. Instead, it was decided to restrict the Phase III treatment to the first two missions and use the third mission for Phase IV. Three factors motivated this decision. First, the pilot evaluation showed that subjects required much more time than had been anticipated for each mission. Second, the Unreal Tournament software on which FixIt depends often crashed when FixIt ran for longer than ten minutes. Although the developer entertained a number of theories for what could be causing this, and tried a number of workarounds including some that seemed to greatly diminish the frequency with which the intermittent, non-deterministic, symptom-free crashes occurred, it still seemed too big of a risk to perform the evaluation without a sub-ten minute time limit to ensure that each subject was able to use the system for the same amount of time.

**Phase IV** Performance Evaluation: In Phase IV all three treatment groups were treated the same, in that Annie offered no help for that segment of the evaluation. As mentioned above, Phase IV applied to the third mission of FixIt.

**Phase V** Post-game: Immediately following the FixIt game, subjects were given a post-game survey to assess subject understanding of domain, as well as their perceptions of the assistance provided in the game and the overall difficulty of their tasks.

The primary goal of the main evaluation is to test whether or not the automated scaffolding produced by Annie is useful help to subjects in the exploratory learning environment. Two measurable indicators of usefulness are the number of discrete learning tasks each subject completes, and the amount of time it takes each subject to complete the same set of tasks.
By separating the experimental subjects into three treatment groups, where the only difference between the groups is how much Annie is allowed to help, there is an opportunity to answer two types of questions for each hypothesis. First, the results can be analyzed to detect statistically significant differences between the three discrete categories of subjects. For instance, did the subjects in the group that received an ablated level of Annie’s interventions perform differently than those in the other two groups? Second, because group membership implies the degree to which Annie is being applied to that particular subject’s experience, questions can be answered about the degree to which Annie’s involvement is useful.

Thus, with two primary measures, and two methods for measuring them, the main study tested four main hypotheses:

**Hypothesis 1A: Task Completion - Correlation** *The degree to which Annie is allowed to provide help to a subject will be correlated with the number of learning tasks that the subject completes.*

**Hypothesis 1B: Task Completion - Group Differences** *The mean number of learning tasks completed by the group of subjects receiving the full Annie treatment will exceed that of the group receiving the ablated Annie treatment which will exceed that of the control group.*

**Hypotheses 2A: Challenge Completion Time - Correlation** *The degree to which Annie is allowed to provide help to a subject will be inversely correlated with the amount of time it takes a subject to complete each learning challenge. Separate tests are computed for completion times of each of the three sequential learning challenges.*

**Hypotheses 2B: Challenge Completion Time - Group Differences** *The group of subjects receiving the full Annie treatment will complete each learning challenge in less time than it...*
takes for the group receiving the ablated Annie treatment who will take less time than the control group. Separate tests are computed for completion times of each of the three sequential learning challenges.

Data beyond that required to evaluate the four primary hypotheses was also collected. As shown in Appendix C, the written assessments completed by each subject before and after playing the game included questions to gauge subject understanding of how malware interacts with the operating system at a process level and to characterize subject perceptions of FixIt and Annie. A secondary hypothesis of the main evaluation was that Annie would impact post-test gains on these knowledge assessment questions. This hypothesis is deemed secondary because the combination of the short duration of the learning experience and the coarse-grained assessment potential of just a few relevant questions on the survey.

**Hypothesis 3** The degree to which Annie is allowed to provide help to a subject will be correlated with the measures of pre-test to post-test learning gains.

### 5.4.2 Main Evaluation: Results

At the start of each experimental session, Annie randomly selected the treatment group for each subject, resulting in the distribution of subjects to treatment groups shown in table 5.2.

<table>
<thead>
<tr>
<th>Treatment Code</th>
<th>Treatment Description</th>
<th>Number of Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Control: Annie provides no help</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>Ablated Annie: Limited help</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Full Annie: No limitations on amount of help</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>28</strong></td>
</tr>
</tbody>
</table>
To test Hypotheses 1A and 1B, a measure of the percentage of learning tasks completed by each subject was derived from the information available in the FixIt game logs. If a subject completed all three learning challenges, the learning task completion was set to 100. If the subject completed the first two learning challenges, the score was set to 70, and if the subject completed only the first challenge, the score was set to 40. Incremental points were given for partial completion of learning tasks. For partial completion of learning challenge one, five points each were given for picking up the Inspection tool, picking up the Nice tool, and using the Inspection tool or Nice tool on a process panel. For partial completion of learning task two, ten additional points were given for picking up the Destroy tool. For partial completion of learning task three, five additional points were awarded if the subject destroyed the malware parent process, by five more if the subject destroyed the child process, and ten more if the subject destroyed the parent process infecting file on disk. Table 5.3 presents the task completion percentage derived from the FixIt logs for each subject.

To test Hypothesis 1A (Task Completion - Correlation), the treatment code field was treated as a continuous variable depicting the degree to which Annie was allowed to help the subject. The Pearson product-moment correlation coefficient $r$ between the Annie treatment condition and the task completion percentage was $r = 0.68752$ at a significance level of $p < 0.0001$. This means that Hypothesis 1A was accepted, as a high level and statistically significant correlation was found between the degree to which Annie helped a subject and the number of learning tasks that subject completed.

Although the results are quite strong, a caveat must be proffered concerning their interpretation. Given that the range of this variable was restricted to 0, 1 or 2, it means that this statistical analysis treated the Ablated Annie treatment condition as being exactly 50% (a “1” as com-
Table 5.3: Student Task Completion Percentages

<table>
<thead>
<tr>
<th>Subject ID</th>
<th>Treatment Code</th>
<th>Task Completion Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject 1</td>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>Subject 2</td>
<td>2</td>
<td>75%</td>
</tr>
<tr>
<td>Subject 3</td>
<td>0</td>
<td>70%</td>
</tr>
<tr>
<td>Subject 4</td>
<td>1</td>
<td>60%</td>
</tr>
<tr>
<td>Subject 5</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>Subject 6</td>
<td>2</td>
<td>80%</td>
</tr>
<tr>
<td>Subject 7</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>Subject 8</td>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>Subject 9</td>
<td>2</td>
<td>80%</td>
</tr>
<tr>
<td>Subject 10</td>
<td>0</td>
<td>40%</td>
</tr>
<tr>
<td>Subject 11</td>
<td>1</td>
<td>40%</td>
</tr>
<tr>
<td>Subject 12</td>
<td>0</td>
<td>20%</td>
</tr>
<tr>
<td>Subject 13</td>
<td>2</td>
<td>60%</td>
</tr>
<tr>
<td>Subject 14</td>
<td>0</td>
<td>20%</td>
</tr>
<tr>
<td>Subject 15</td>
<td>2</td>
<td>75%</td>
</tr>
<tr>
<td>Subject 16</td>
<td>0</td>
<td>80%</td>
</tr>
<tr>
<td>Subject 17</td>
<td>1</td>
<td>60%</td>
</tr>
<tr>
<td>Subject 18</td>
<td>0</td>
<td>10%</td>
</tr>
<tr>
<td>Subject 19</td>
<td>1</td>
<td>70%</td>
</tr>
<tr>
<td>Subject 20</td>
<td>1</td>
<td>80%</td>
</tr>
<tr>
<td>Subject 21</td>
<td>1</td>
<td>70%</td>
</tr>
<tr>
<td>Subject 22</td>
<td>2</td>
<td>100%</td>
</tr>
<tr>
<td>Subject 23</td>
<td>0</td>
<td>10%</td>
</tr>
<tr>
<td>Subject 24</td>
<td>2</td>
<td>75%</td>
</tr>
<tr>
<td>Subject 25</td>
<td>1</td>
<td>75%</td>
</tr>
<tr>
<td>Subject 26</td>
<td>0</td>
<td>20%</td>
</tr>
<tr>
<td>Subject 27</td>
<td>2</td>
<td>75%</td>
</tr>
<tr>
<td>Subject 28</td>
<td>2</td>
<td>75%</td>
</tr>
</tbody>
</table>

pared to a “2”) as Annie-intensive as the Full Annie treatment condition. There is no way to know, for a given subject receiving the Ablated Annie treatment, whether the Full Annie would have provided twice as much help, it is simply the case that the Full Annie treatment had the potential to provide that much help, since it allowed remediations to be repeated.
To test Hypothesis 1B (Task Completion - Group Differences), the General Linear Model procedure was used to analyze the variance (ANOVA) between the treatment groupings and the task completion percentages. This ANOVA revealed a highly significant effect for treatment, $F(2, 25) = 11.26, (p < 0.0003)$. Because this result was statistically significant, a post-hoc Student-Newman-Keuls mean comparison test was run to find out which mean differences were responsible for the effect. This test found statistically significance differences for the mean task completion percentages between each the three treatment groups (34.44, 55, and 79.5). Together, these tests provide strong evidence for Hypothesis 1B, that the group of subjects receiving Full Annie treatment outperformed those in the Ablated Annie group, who in turn outperformed the No Annie control group.

The measure used in testing each variant of Hypotheses 2A and 2B was simply the number of seconds required to complete each of the three major learning challenges. These measurements are tabulated in table 5.4. An important caveat to note here is that due to a problem discovered in the Unreal Engine underlying the FixIt game, all game sessions were automatically terminated after eight minutes of game time. The reasons for this are described in detail in section C.2 of Appendix C. An unfortunate consequence is that for learning challenges not completed by the end of the eight-minute time limit, a maximum value of 480 seconds was recorded. Clearly, this is inaccurate, but it is inaccurate in the direction that actually makes it more difficult to obtain the variants of Hypothesis 2. It introduces a “floor effect” that skews the poorer performances down to an eight minute maximum, where without a time limit it could have taken twenty minutes or an hour to actually complete the challenge.

To test Hypothesis 2A (Challenge Completion Time - Correlation), the treatment code field was again treated as a continuous variable depicting the degree to which Annie was allowed
<table>
<thead>
<tr>
<th>Subject ID</th>
<th>Challenge 1 Time</th>
<th>Challenge 2 Time</th>
<th>Challenge 3 Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject 1</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 2</td>
<td>161</td>
<td>254</td>
<td>480</td>
</tr>
<tr>
<td>Subject 3</td>
<td>266</td>
<td>331</td>
<td>480</td>
</tr>
<tr>
<td>Subject 4</td>
<td>155</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 5</td>
<td>418</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 6</td>
<td>82</td>
<td>257</td>
<td>480</td>
</tr>
<tr>
<td>Subject 7</td>
<td>59</td>
<td>93</td>
<td>343</td>
</tr>
<tr>
<td>Subject 8</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 9</td>
<td>83</td>
<td>106</td>
<td>480</td>
</tr>
<tr>
<td>Subject 10</td>
<td>292</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 11</td>
<td>251</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 12</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 13</td>
<td>301</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 14</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 15</td>
<td>253</td>
<td>325</td>
<td>480</td>
</tr>
<tr>
<td>Subject 16</td>
<td>131</td>
<td>300</td>
<td>480</td>
</tr>
<tr>
<td>Subject 17</td>
<td>57</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 18</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 19</td>
<td>176</td>
<td>332</td>
<td>480</td>
</tr>
<tr>
<td>Subject 20</td>
<td>194</td>
<td>284</td>
<td>480</td>
</tr>
<tr>
<td>Subject 21</td>
<td>208</td>
<td>320</td>
<td>480</td>
</tr>
<tr>
<td>Subject 22</td>
<td>26</td>
<td>80</td>
<td>374</td>
</tr>
<tr>
<td>Subject 23</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 24</td>
<td>94</td>
<td>137</td>
<td>480</td>
</tr>
<tr>
<td>Subject 25</td>
<td>165</td>
<td>382</td>
<td>480</td>
</tr>
<tr>
<td>Subject 26</td>
<td>480</td>
<td>480</td>
<td>480</td>
</tr>
<tr>
<td>Subject 27</td>
<td>250</td>
<td>376</td>
<td>480</td>
</tr>
<tr>
<td>Subject 28</td>
<td>164</td>
<td>317</td>
<td>480</td>
</tr>
</tbody>
</table>

to help the subject. The Pearson product-moment correlation coefficient $r$ between the Annie treatment condition and the time taken to complete each of the three learning challenges was computed. The results of these three correlation computations are shown in table 5.5. As depicted in the table, some evidence was found to support hypotheses 2A: a strong and significant
correlation was found between increased help from Annie and decreased time required to complete challenges 1 and 2. For learning challenge 3, however, a weak negative correlation was found at a marginal level of significance. As seen in table 5.4, only two subjects completed the third challenge, both received the Full Annie treatment, and both finished toward the end of the available time allotment. This paucity of successful completions combined with the diluting floor effect of a 480 second maximum for unsuccessful completions reduces both the strength of the negative correlation and its significance.

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Correlation Coefficient</th>
<th>p-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds to complete Learning Challenge 1</td>
<td>$r = -0.64$</td>
<td>$p &lt; 0.0002$</td>
</tr>
<tr>
<td>Seconds to complete Learning Challenge 2</td>
<td>$r = -0.63$</td>
<td>$p &lt; 0.0003$</td>
</tr>
<tr>
<td>Seconds to complete Learning Challenge 3</td>
<td>$r = -0.32$</td>
<td>$p &lt; 0.09$</td>
</tr>
</tbody>
</table>
To test Hypotheses 2B (Challenge Completion Time - Group Differences), the General Linear Model procedure was used to analyze the variance between the treatment groupings and the task completion percentages. The tabulated ANOVAs are shown in table 5.6. The ANOVA results mirror those of the correlation comparisons, in that statistically significant differences between treatment groups was observed for the completion times of learning challenges 1 and 2, but not for learning challenge 3.

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>ANOVA Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds to complete Learning Challenge 1</td>
<td>$F(2, 25) = 9.20, \ (p &lt; 0.001)$</td>
</tr>
<tr>
<td>Seconds to complete Learning Challenge 2</td>
<td>$F(2, 25) = 10.73, \ (p &lt; 0.0004)$</td>
</tr>
<tr>
<td>Seconds to complete Learning Challenge 3</td>
<td>$F(2, 25) = 1.97, \ (p &lt; 0.1607)$</td>
</tr>
</tbody>
</table>

As with hypothesis 1B, post-hoc Student-Newman-Keuls tests were run to determine which group mean differences were responsible for the significantly different ANOVAs. Unlike hypothesis 1B, however, where each of the three groups was found to have significant pair-wise differences with the other two groups on the task completion percentage measure, the differences on the challenge completion time measure of hypothesis 2 were less distinct. The Student-Newman-Keuls test on hypothesis 2B for Learning Challenge 1 found statistically significant pairwise-differences in the mean completion times between the control group (No Annie) and the means of each of the two groups that received help from Annie. However, statistically significant mean differences were not seen between the Ablated Annie and Full Annie groups on this measure.
In evaluating hypothesis 2B for learning Challenge 2, the statistically significant mean differences between the Full Annie group as compared to the Ablated Annie and No Annie control groups, but no significant difference between the mean times for the Ablated Annie and No Annie control groups. One problem with this hypothesis is that approximately half of the subjects did not finish learning Challenge 2, so the floor effect of 480 seconds is again diluting the statistical differences between groups. What can be said is that evidence was found to partially support Hypotheses 2B for learning challenges 1 and 2, but not for learning Challenge 3.

Hypothesis 3 is that Annie will have a positive impact on pre-test to post-test gains on the subjects’ written answers to the malware knowledge assessment questions. As expected, insufficient evidence was produced to accept or reject this hypothesis, probably due to the short duration of the learning experience, the coarse-grained assessment potential of just a few relevant questions on the survey, and the low degrees of freedom from having fewer than 12 subjects per group. A weak, but statistically insignificant positive correlation \( r = 0.303, p < 0.1167 \) was found between the degree of Annie treatment and the pre-post learning gain measure. Some argue that when a \( p \)-value is not less than 0.1, analysts should not even report the correlation coefficient. In this case, it is reported anyway, because it had expected there would be insufficient data to register a strong finding, and the correlation, though weak, at least leans in the expected positive direction.

5.4.3 Main Evaluation: Conclusions

The results of the main evaluation show that the automatic scaffolding generated by Annie helped subjects complete more tasks and complete those tasks in less time. Statistically strong support was found to accept Hypothesis 1A: increased intervention from Annie was correlated with increased number of learning tasks completed. Hypothesis 1B also found support as the
full Annie group outperformed the Ablated Annie group who outperformed the Control group on the measure of task completion, with statistically significant mean differences between the three treatment groups.

Support was also found for Hypotheses 2A and 2B in that Annie would help subjects finish the first two learning challenges faster and that the group-wise differences in completion speed were be significant. Insufficient evidence was found to support hypothesis 2A and 2B for learning challenge 3, but this is likely to have been a result of the time-limit placed on FixIt and the fact that only two subjects were able to complete the entire set of learning tasks.

Insufficient evidence was found to either support or reject Hypothesis 3, that Annie would have a positive impact on pre-test to post-test gains. Improvements to the learning environment in a future experiment may be sufficient to push the evidence into the statistically significant range, without requiring changes to Annie.
The thing about Annie

This pilot study is focused on how accurate my system can guess what you know about how the game works. I believe the real authority on that subject is you, so I would like you to answer the following questions as accurately as possible so I can assess the accuracy of the guesses made by my software.

1. How many different “tools”/guns did you pick up? ______
2. Did you see the “droids” (box shaped robots that move across the floor).  Y___ N___
3. Did you see the “process panels” (translucent panels arranged in an outward-facing circle).  Y___ N___

For each weapon/tool you fired on a droid, what effects did you observe AND RECOGNIZE AT THE TIME (as opposed to realized after reading the choices here)? (Circle all that apply)

4. Weapon One – Purple laser shock rifle (Left Mouse Button):
   A) Nothing   B) Moved it   C) Stopped it   D) Destroyed it   E) Slowed it down   F) Speeded it up

5. Weapon One (Right Mouse Button):
   A) Nothing   B) Moved it   C) Stopped it   D) Destroyed it   E) Slowed it down   F) Speeded it up

6. Weapon Two – Red Rockets (Left Mouse Button):
   A) Nothing   B) Moved it   C) Stopped it   D) Destroyed it   E) Slowed it down   F) Speeded it up

7. Weapon Two (Right Mouse Button):
   A) Nothing   B) Moved it   C) Stopped it   D) Destroyed it   E) Slowed it down   F) Speeded it up

Figure 5.5: Pilot Evaluation Self-Assessment (page 1)
Chapter 6

Limitations and Future Work

As described in the preceding chapters, this dissertation makes several significant research contributions. However, there are a number of limitations in the current implementations of both Annie and FixIt that should be highlighted. Several proposed extensions to Annie’s capabilities which address its current limitations are discussed below.

This chapter is divided into four sections. Section 6.1 describes limitations of the current implementation of Annie. Section 6.2 describes limitations of the FixIt evaluation environment. Section 6.3 describes proposed expansions of Annie’s capabilities. Finally, section 6.4 describes experiments to aid in evaluating Annie’s existing and expanded capabilities.

6.1 Limitations of the Annie System

Annie was designed to be domain-independent, but has been evaluated in just a single domain. Annie’s processing is free of explicit links to the FixIt game beyond the knowledge of plan operators provided as input. Nevertheless, to support the claim that Annie is domain-independent, future work should evaluate Annie in domains and learning environments that
differ from FixIt.

An implicit limitation of Annie is that it depends upon accurate task-based descriptions of the domain and learning objectives. In addition, Annie requires a game world with which it communicates using these task-based descriptions, and it requires that the game world is instrumented to communicate with Annie and carry out the actions Annie recommends. For example, Annie requires the game world to implement a method to generate a text message for display to the subject as a result of Annie sending a remediation message to show text to the student.

A consequence of Annie’s dependence on task-based knowledge representation is that Annie is not a good choice for teaching knowledge that is independent of the relationships between tasks, for example, a broad compendium of facts unrelated to process or procedure. A second consequence of Annie’s dependence upon plan-based reasoning is that it will perform best in an environment that requires open choices among a set of related actions. A knowledge base that reflects this type of environment is not always as easy to provide as it might seem. Care should be taken to ensure that the particular $ZLib$ representing the domain affords meaningful alternative success paths to better test student understanding of the tasks in the domain.

One of the most significant limitations of Annie’s current design is its use of an overly simplistic model of student belief about the task domain. The current enumeration of a set of linearly-ordered belief levels and the simple method for modeling change in belief between them restricts Annie’s ability to realistically model the way that a student changes his or her beliefs over time. Further, the coarse-grained model of belief likelihoods prevents Annie from considering subtle remediation strategies that could be particularly effective.
6.2 Limitations of the FixIt Evaluation Environment

FixIt does not yet provide as rich a set of tutorial actions as needed to exercise all of the remediation capabilities Annie could offer. Currently, it allow for text-based prompts, text-to-speech translation, and a limited facility to show features of the learning environment through pre-scripted camera control. An important capability would be for FixIt to direct non-player controlled characters to perform Annie-specified actions in-game as a method of demonstrating those actions to the student. A precedent for this type of direction is found in Steve [68] system described in section 2.2.1.

6.3 Future Work: Expanded Capabilities

This section describes substantial expansions to Annie’s capabilities that are strong candidates for future work. Many of the capabilities to be expanded are described in detail in Chapter 4.

6.3.1 Improved Representation and Reasoning for Student Beliefs

As mentioned above in section 6.1, a significant limitation of Annie’s current design is its use of an overly simplistic model of likelihoods of particular student beliefs about the task domain. Changing this representation from a small set of discrete categories to the much more common model of belief likelihood by a numeric range over the interval from zero to one would confer several advantages. First, it would allow for more precise estimates of belief likelihood at a particular point in time. Second, it would aid in remediation selection, as a broader range of belief likelihoods should make it less likely that multiple beliefs share the same likelihood estimate. Third, it would allow for more precise adjustment of belief likelihoods when diagnostic criteria are employed.
There are a number of examples of the use of such models in systems similar to Annie. Bunt and Conati [10] use Bayesian Belief Networks (BBNs) [18] to model student knowledge in a math domain. Mott [49] uses a Dynamic Decision Network (DDN) to model student knowledge in the domain of microbiology. Albrecht and Zukerman [1] use BBNs for student plan recognition in an adventure game.

6.3.2 Improved Support for Character-Based Remediation

The current implementation relied largely on text-based remediations common in traditional Intelligent Tutoring Systems. Much of the potential for new modes of tutorial expression in such environments is not yet harnessed by Annie. For example, game-based learning environments can be populated with multiple pedagogical agents who can demonstrate tasks, manipulate objects in the environment and interact with the student in ways that both instruct and challenge learners.

For example, one of the first serious applications of a game-based learning environment was Steve [68] described in section 2.2.1. Steve is an animated pedagogical agent who teaches human students to operate the engines of a naval surface ship in a virtual environment through demonstration of the required procedures. To fully leverage the power of immersive 3-D environments in this manner, Annie requires a more comprehensive set of general remediations to control the behavior of non-player characters.

6.3.3 Expanded Task-Based Tutorial Authoring Tool Set

Wide dissemination of Annie will not be possible without a powerful and useful tool set to assist tutorial authors. Several recent projects have shown promise in aiding in the construction and debugging of plan libraries [79, 80, 73, 45, 13]. Bowman [79, 80] and Wide-Ruled [73],
two systems that support human authoring of plan libraries for narrative environments, suggest methods to harness modern graphical user interfaces to aid in the plan-library construction. GIPO [45] provides authors with tools to help visualize hierarchical task models, and Bowyer [13] defines methods to bridge the declarative representations of planning with the procedural effectors of action in game domains.

Integrating aspects of these technologies to aid in the external editing of tutorial domains, as well as the diagnostic and remediation templates to be applied in those domains, could greatly ease the burden of authoring new tutorial domains for Annie.

### 6.3.4 Increased Expressivity of Plan-Based Reasoning

In the current implementation, Annie reasons about the preconditions and effects associated with particular operators, but does not extend that reasoning to the level of the bindings of individual variables appearing in terms in preconditions and effects. Such reasoning would add significant expressive power to Annie. First, it would allow Annie to differentiate between different instances of the same operator when applied to two different objects in the domain. Second, it would allow Annie to provide more specific remediations that would be associate with the particular objects or subclasses of objects to which the application of a particular operator is most relevant. Third, incorporating this level of reasoning into diagnostic and remediation templates would increase their ability to discriminate between different task contexts. For example, knowing the particular object bound to a particular precondition in a failed action attempt could allow identification of that object to be passed through a remediation template to create a more tailored remediation in the learning environment.
6.4 Future Work: Proposed Experiments

The main evaluation of Annie described in this dissertation compared the effectiveness of Annie’s scaffolding to a control treatment that offered no scaffolding at all. While the results suggested that Annie’s scaffolding was helpful, a more substantive evaluation would be to compare Annie to alternatives that are capable of providing similar forms of assistance. A possible experimental design would be to incorporate into the evaluation environment all the forms of assistance Annie generates over a series of trial runs. If these were then deployed randomly during the actual evaluation where the amount and pacing of the scaffolding were set to emulate those of Annie, it would provide a control group that would specifically evaluate the value of Annie’s reasoning about the timing and selection of particular learner scaffolds.

One of the strengths Annie should derive from its plan-based reasoning is to handle learning situations in which a student is presented with multiple simultaneous choices that can lead toward successful goal completion. A 2x2 Latin-Square evaluation could be run that varies the number of simultaneous choices given to two different groups of subjects, and compares Annie’s performance between those two groups to that of a capable control treatment like that described in the previous paragraph. Differences between these four treatment groups could show how Annie’s effectiveness varies with increased complexity of task choices, and compare that variance with a control. The hypothesis to be tested is that the effectiveness of Annie compared to the control group would increase with increased numbers of simultaneous task choices.

In the current experimental design, care was taken to orient the student to the associations between what was being emulated in the learning environment and the concepts they represented in the learning domain. Nevertheless, in observing the interactions of subjects with the
environment, numerous instances of student confusion were noted due to the weak semantic connection between an action occurring in the game and the operating system or malware concept that the action was supposed to represent to the student. Although this separation is likely to be common in simulation environments, it is not accounted for in Annie’s reasoning. It is possible that the degree to which the student understands game semantics influences learning more than the specific cognitive gaps that Annie is concerned with filling. To better understand this effect, an evaluation could vary the degree to which students are oriented to the symbols and meanings of the environment to compare the size of this effect on learning vs. the size of effect seen due to relative intelligence the automated tutor.

Learning technologies are often evaluated for their effect on a learner’s retention of acquired knowledge. Retention can be tested by evaluating how well a student performs in subsequent learning sessions, often separated by a few days or weeks. The possibility of performing multiple time-separated evaluations on the same student begs the question of whether Annie should be extended to allow the saving and restoration of a student model associated with a particular student. If so, an evaluation to determine the effect of Annie on retention could be combined with an evaluation comparing Annie’s effectiveness when using learning models carried over from one session to the next to its effectiveness when models are discarded after each session.
Chapter 7

Conclusion

The goal of the work described in this dissertation is to develop a new conceptual framework within which systems can be implemented that integrate the capabilities of intelligent tutoring systems within non-linear, exploratory digital games for learning. A system adhering to this framework should be domain and game-independent, but capable of automatically generating customized help, or scaffolding, based on the needs of individual learners at run-time. The concepts constituent to this framework were implemented in a system called Annie, and the effectiveness of that system was formally evaluated in two separate experiments with human subjects. The evaluation showed that Annie was successful in providing useful guidance to learners and may be capable of increasing student’s learning gains.

This work is applicable across a wide range of domains because its core abstractions are the actions the student or the system can use to change the state of the environment. Because it focuses on actions change the states of objects in a particular domain, it is expected to be most effective in domains where tasks typically feature complex hierarchies of inter-related actions. Scientific discovery learning in domains such as RNA transcription, photosynthesis,
thermodynamics or the Krebs cycle that are good examples of domains where this system is expected to best complement traditional modes of instruction.

The dissertation describes in detail how Annie builds a student model based on the individual student’s sequence of action attempts, and how it introduces new plan reasoning algorithms to manage the uncertainty of what the student knows and what the student is trying to do. Annie employs this reasoning in choosing when, what, and how to raise or lower the level of learning challenges presented to the student through the exploratory game paradigm. The key contribution of this work is how Annie adapts – or scaffolds – the learning challenges within independently developed exploratory learning environments.

An exploratory learning environment called FixIt, situated in the domain of computer malware as viewed at level of the operating system, was created for the purpose of evaluating Annie’s effectiveness. Two formal evaluations of Annie and FixIt were performed with human subjects. The results of the pilot evaluation showed that Annie’s plan-based knowledge representation was a useful basis for modeling student knowledge. Annie’s estimations of what each subject knew about the environment was compared to student self-assessments. The same comparison was made between a specially trained human observer and the student self-assessments. Annie’s overall accuracy was found to be equivalent to that of the human expert at a high level of statistical significance, and a strong correlation was found between Annie’s subject-by-subject accuracy and the accuracy of the human expert in modeling each subject’s domain knowledge.

The results of the main evaluation showed that the automatic scaffolding generated by Annie helped subjects complete more tasks and finish those tasks they did complete in less time. Statistically strong support was found for the hypothesis that increased intervention from Annie
was correlated with increased number of learning tasks completed. The experiment also found statistically significant mean differences in learning task completion between a control group, a group of subjects who received only partial help from Annie, and a group that received the full measure of Annie’s assistance. The main evaluation also showed that increased levels of intervention by Annie decreased the time required to complete the two core learning challenges confronted by the subjects. Some evidence was provided in support of Annie having a positive effect on learning gains, but this evidence was not at a statistically significant level.

As noted in the detailed design, there are opportunities for future work to extend the depth and complexity of Annie’s reasoning in several areas. Although it is encouraging that Annie was shown to be effective in the initial evaluation, it is likely that its effectiveness can be increased. First, the student model currently relies on a coarse-grained estimate of the likelihood the student knows each particular fact in the plan-based description of the world. Substituting a Bayesian Belief Net implementation may improve Annie’s remediation strategies. Second, Annie’s strategy for selecting which beliefs should be remediated next is currently weighted toward proximal actions. It is possible that a strategy weighted toward more important or critical actions may have benefits.

In summary, the system implemented for this dissertation performed well against its goals of automatically generating scaffolding to guide students in exploratory learning environments. Through the development and evaluation process, many interesting avenues have been identified for future investigations of the range of applicability and effectiveness of enhancements to Annie and the framework it implements.
REFERENCES


APPENDICES
Appendix A

Evaluation Environment Details

An abbreviated description of the FixIt evaluation environment can be found in Chapter 5, which begins on page 88. This appendix provides a more detailed and complete description. This appendix is divided into seven sections. The first five sections describe general features of FixIt that could be made available in any evaluation. The sixth section delineates the specific modifications of the environment made for the pilot evaluation. The final section describes the modifications made for the main evaluation.

A.1 FixIt: Level Design “Geography”

FixIt, the initial evaluation environment for Annie, is an immersive 3-D virtual world built on the Unreal Tournament 3 engine manufactured by Epic Games. To build this environment requires two separate undertakings: level design and scripting. Level design is the process of creating the map of 3-D scene geometry that bounds the environment and then populating it with graphic textures, objects, sounds, lighting, and animations. The level designer specifies all the visible aspects of the map, and also implements the invisible functions that control where
the player can or cannot walk, which objects are movable or mutable, the places where actors \textit{spawn}, and the paths taken by dynamic actors in the environment.

Several design goals guided the design of the FixIt level. The preeminent goal was to establish that the environment was an abstract representation of the inside of a fictional computer. The “floor” of the environment was depicted as a semi-translucent dappled green chosen to match that of typical circuit boards. Highly reflective platinum colored channels were cut into the circuit board to represent electrically conductive traces between components. To match
A second level design goal was to make it clear which components of the computer were being modeled and which pathways led to each component. To achieve this goal, generic (intentionally textured in with nondescript white material) 3-D models representing each component were added just above and beyond the grey walls that demarcate the walkable area of the level. Each model was placed at the junction where the pathway to the corresponding component ends terminates in a conduit that carries the messengers through the wall. Several of these can be seen in the screen shots of Figure A.1 shows birds-eye view in A.1a. In addition, a sign with the name of the component was placed on a gate-like arch at the termination point of the messaging pathway.

The CPU of the computer is found in the center of the circuit board, just below the point where the student’s avatar spawns. The centrality of this location was chosen to keep the student focused on the actions that demonstrate the core concepts taught in the game. Also, a circular design was chosen to simplify the student’s task of seeing the traffic flows between the CPU and the other components of the computer. The key features of the CPU area are the process panels, and the system monitor panel.

The second large area of the level map is the disc drive. The floor of the disc drive area is a metallic circular texture, in contrast to the green circuit board. A hierarchical file system is depicted by semi-circular rooms with translucent walls that are nested inside each other to represent each directories and sub-directory. Each directory-room features a prominent sign, and files are shown as oblong electronic boxes on the floor of the enclosing directory. The
connection between the disc drive and the circuit board is an elevated walkway that passes through an interior wall to make it clear to the student that the disc drive is a separate entity. Because the pilot evaluation showed that it was taking too much time for the students to cover the distance of the walkway, a “hover board” was added that the student automatically mounts when stepping on to the walkway and dismounts when leaving it, as the hover board travels two to three times faster than the player can walk.

Other important components of the level design are the restart/restore control panel found on the platform above the CPU, and the elevator that brings the student down to the CPU platform. The elevator was designed to meet several purposes. First, the player spawns onto a torus the encircles the elevator platform itself. No actions are triggered in the game until the player is able to use the navigation keys to move the avatar from the encircling disc onto the circular platform. A cylindrical blocking volume keeps the player from moving off the outside of the torus. Once the player has moved safely onto the disc, it begins a descent onto the platform above the CPU, giving the student a 360 degree top-down view of the environment in the process. Thus, the student needs to do just a single, simple action (moving onto the platform) and then must simply observe the environment for the several seconds or so it takes for the elevator to descend to the platform. This helps establish with the student the requirement to interleave action with observation of the environment.

### A.2 FixIt: Actors

The two most important non-player actors in the game are the interdependent process panels and mouse droids. Each process has a distinct name, graphical icon, and color, shown on the translucent panel in the CPU hub that corresponds to that process. As processes send and re-
To receive messages to various parts of the computer, the packets comprising these messages are depicted as box-shaped *mouse droids* matching the color and iconography of the corresponding process that emerge from underneath the corresponding panel (if outgoing from the CPU) or at the corresponding system component (if incoming toward the CPU). Mouse droids slide along the platinum-colored circuit pathways following prescribed multi-point paths between the CPU and other components of the system to provide visual indications of the direction of messaging traffic generated by each process. The volume of mouse droids moving along particular pathways provides a visual indicator of the amount of work being done by a particular process.

**A.2.1 FixIt Actors: Processes**

The FixIt implementation of processes is quite flexible. The only fixed attributes of a particular process are its visual icon, color, and label text, shown in Table A.1. The process owner (system vs. user) is defined programmatically but is included in the table to illustrate that all system processes use the gear icon except for “Fluffy Bunny”, which is intended to be treated as a malware process that illicitly masquerades as a system process. Run-time programming controls which processes are to be instantiated, the locations of each process panel, as well as the frequency with which mouse droids are requested and the paths the droids travel. There are sixteen different types of processes and sixteen slots for panels, but the programmer can specify the spawning of zero or more instances of any process and request specific, as well as fixed or random panel slot assignments. Figure A.2a shows adjacent process panels for the Syswatch.exe and MBox.exe processes. The translucent white panel to the right of MBox.exe is an unassigned panel slot.
Figure A.2: FixIt: Important Actors

(a) Process Panels (SysWatch.exe and MBox.exe)

(b) Mouse Droid (IWeb.exe)
<table>
<thead>
<tr>
<th>Label Text</th>
<th>Icon</th>
<th>Color</th>
<th>Expanded Name</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNt.exe</td>
<td>Gear</td>
<td>Blue</td>
<td>Connect to Network</td>
<td>System</td>
</tr>
<tr>
<td>DtBack.exe</td>
<td>Gear</td>
<td>Yellow</td>
<td>Data Backup</td>
<td>System</td>
</tr>
<tr>
<td>Explorer4p.exe</td>
<td>Gear</td>
<td>Orange</td>
<td>Explorer</td>
<td>System</td>
</tr>
<tr>
<td>Index.exe</td>
<td>Gear</td>
<td>Bronze</td>
<td>Indexer</td>
<td>System</td>
</tr>
<tr>
<td>Logon.exe</td>
<td>Gear</td>
<td>Violet</td>
<td>Login</td>
<td>System</td>
</tr>
<tr>
<td>PlugMng.exe</td>
<td>Gear</td>
<td>Magenta</td>
<td>Plug-in Manager</td>
<td>System</td>
</tr>
<tr>
<td>SysWatch.exe</td>
<td>Gear</td>
<td>Purple</td>
<td>System Watch</td>
<td>System</td>
</tr>
<tr>
<td>FBun.exe</td>
<td>Bunny</td>
<td>Green</td>
<td>Fluffy Bunny</td>
<td>System</td>
</tr>
<tr>
<td>IWeb</td>
<td>Globe</td>
<td>Red</td>
<td>iWeb Browser</td>
<td>User</td>
</tr>
<tr>
<td>KeyIO.exe</td>
<td>Skeleton Key</td>
<td>Aqua</td>
<td>Key Logger</td>
<td>User</td>
</tr>
<tr>
<td>Lbug.exe</td>
<td>Ladybug</td>
<td>Vermillion</td>
<td>Ladybug</td>
<td>User</td>
</tr>
<tr>
<td>MBox.exe</td>
<td>Envelope</td>
<td>Pale Green</td>
<td>Mailbox</td>
<td>User</td>
</tr>
<tr>
<td>Mplay.exe</td>
<td>Musical Notes</td>
<td>Rose</td>
<td>Music Player</td>
<td>User</td>
</tr>
<tr>
<td>Ofw.exe</td>
<td>Document</td>
<td>Silver</td>
<td>Office Work</td>
<td>User</td>
</tr>
<tr>
<td>Smax.exe</td>
<td>Shield</td>
<td>Emerald</td>
<td>Security Max</td>
<td>User</td>
</tr>
<tr>
<td>WoodenBadgerTrojan.exe</td>
<td>Horse</td>
<td>Grey</td>
<td>Wooden Badger</td>
<td>User</td>
</tr>
</tbody>
</table>

A process manager limits the total population of mouse droids active in the system at any one time. Each process must request permission from the process manager to spawn a new mouse droid. The frequency with which each process makes these requests is programmatically defined. When the process manager grants permission to a given process, it sends that process a spawn voucher message, but the process is not allowed to redeem the voucher until it becomes active. Process activation is visually depicted by a brightly animated shield effect light that rotates sequentially around the hub to symbolize the round-robin activation of each process. When the process has a redeemable voucher and it is activated, it will spawn a droid. If the mouse droid’s path is outgoing from the CPU, its spawn location is automatically set to the space just below and behind the corresponding process panel.
A.2.2 FixIt Actors: Mouse Droids

The *mouse droids* represent message packets being carried from one part of the computer to another to perform work for a particular process. To help focus the student’s observations on the activities of each process, the mouse droids are the only actors in the environment that move by themselves. The main responsibility of each mouse droid is to follow a pre-defined path described by a sequence of invisible path nodes embedded in the level design. These paths are built at run-time and assigned programmatically to the droids that correspond to given process. Figure A.2b shows a mouse droid for the IWeb.exe process en route from the CPU to the Network.

A.3 FixIt Actors: System Status Designations

The FixIt system is built with visual and auditory indicators of the virtual system status as depicted in Figure A.3. In normal mode (Figure A.3a, the sky dome above the surface of the level is illuminated and appears much like a sunless blue sky.

When a process begins consuming too many resources, (due to a system-initiated trigger), the system enters warning mode which is shown in Figure A.3b. Warning mode is meant to alert the student that something is not working properly. A recorded announcement says “A system slowdown has been reported, please investigate,” or “A network slowdown has been reported, please investigate,” depending on which problem is in effect. The announcement repeats every 15 seconds. Visually, the back-lit illumination that makes the sky dome a pleasant light blue is extinguished and a yellow glow suffuses the environment under a black sky. Warning mode is terminated when the player resolves the problem that is causing the process to misbehave.
When the player destroys a required system program, the system enters “Fail-Safe” mode. Fail-safe mode sends a recorded announcement to let the student know that a restart must be performed to restore full system functionality, and the student is responsible for initiating the restart. The announcement is “The system is now in fail-safe mode. A system restart is required.” Visually, as shown in Figure A.3c, the scene is differentiated from the warning mode by a red suffused glow as opposed to yellow. The only escape from Fail-safe mode is to perform a restart.

Finally, restart mode is invoked when the player uses any tool on the “System Restart Conduit.” An pre-recorded announcement “The system is now in restart mode” plays, and the entire environment is bathed in supersaturated blue, as seen in Figure A.3d, symbolizing the system shutdown that occurs with the infamous “Blue Screen Of Death”. All mouse droids and processes are terminated at the initiation of a restart, but after ten seconds or so the processes start up again and the environment returns to normal mode.

A.4 FixIt: Student Actions and Tools

A.4.1 Investigate Tool

Students have three primary ways to interact with the learning environment. First, they can observe the environment to understand the details of what is happening in a given learning exercise, and generate ideas for actions to be taken. Second, they can move through the environment to gain better perspective or put themselves in position to take particular actions. Occasionally, movement can result in a state change in the world, when the player’s avatar moves over a “pick-up,” a particular location where the character implicitly gains possession of an object at that location. These first two modes of interaction are provided through as is
typical in many computer games, through simultaneous control of keyboard keys for character movement, and the mouse for moving the “camera”, or first person view. The third method for the student to interact with the learning environment is through a set of “tools,” which are basically re-purposed and re-labeled weapons from the Unreal Tournament arsenal. For example, the Information Tool, when fired on a process panel, causes a translucent pop-up window to be rendered on the student’s Heads Up Display, or HUD, that provides a text description of some of the operational details of that process. Similarly, when fired on a mouse droid, the Information Tool shows text describing the source and destination locations of the mouse droid, its owning process, its payload type and a sample of its payload data.
A.4.2 Nice Tool

When applied to a process panel, the nice tool returns its resource consumption to normal levels, in the event the process is not malware and has gone haywire. Otherwise it has no effect. Slows down an individual droid.

A.4.3 Destroy Tool

When the destroy tool is used on a mouse droid it is destroyed and disappears. When used on a process panel, it destroys the process, returning the panel to its blank state.

A.4.4 Speed Up Tool

When the speed tool is used on a mouse droid, the droid moves more quickly. When used on a process panel, it causes the process to increase the rate at which its mouse droids are spawned.

A.4.5 Pause Tool

When the pause tool is used on a mouse droid, the droid stops. It works as a toggle, so that the second use of the pause tool resumes its movement. Similarly, when used on a process panel, the pause tool causes the process to stop or restart spawning of mouse droids.

A.5 FixIt: Heads-Up Display

The HUD is intentionally sparse compared to many game environments. The only persistent displays are the score and the tool selection menu. Informational displays are displayed in the HUD as a result of a student moving the character to touch a kiosk or using the investigate tool on a process panel, mouse droid, or file.
A.5.1 Score

The job of the student is to monitor the system’s health and diagnose any problems through observation of the droids and the process panels. A points-based scoring system was implemented to provide a visual indicator of system health and its effect on player score. Full system health is 100. As problems occur this number decreases. The student receives five points every few seconds when the health is at 100, and fewer points as this decreases. The video explained that the system health score will decrease if the player harms healthy portions of the system. The scoring mechanism was added to discourage students from firing indiscriminately on all processes and droids, and to help focus students’ attention on the goals of the learning challenges, as the system health would be decreased by 25 or 50 points each time a challenge started.

A.5.2 Notes

An additional help mechanism added to FixIt is the placement of informational notes that “pop-up” into the HUD as easy-to-read 2D images. Default contents of each note are specified but can be overwritten by Annie, although Annie has yet to take advantage of this feature.

A.6 FixIt: Pilot Evaluation Modifications

To accommodate the goals of the pilot evaluation, a wall was inserted to cut-off player access to the disc drive, so that the player’s movements would be constrained to a smaller area. Five tools were made available in the environment.
A.7  FixIt: Main Evaluation Modifications

The role of the student in the version of FixIt used in the main evaluation is to find and repair problems in the fictional computer by using the appropriate tools on the appropriate objects and the appropriate times. A sequence of three learning challenges, or missions, guide the student through the learning content. The challenges build on each other so that the student must reuse skills acquired in earlier missions. In general, Annie can benefit from learning challenges that build on earlier challenges, as it extends the relevance of the student model, providing more opportunities for scaffolding and fading than would be possible if the tasks were more discrete.

The first learning challenge involves a runaway process, which can be fixed by the student applying the “Nice” tool to the affected process panel. The second scenario features a SPAM-bot, a malicious program that sends its droids to the system mail program, which obligingly dispatches its own droids carrying the SPAM payload out to the network. The student can solve this problem by using the Delete tool on the SPAM-bot process. Incorrectly using the Delete tool on the mail process will not solve the problem, as the SPAM-bot will continue to send droids to the network disguised as mail droids.

In the third scenario, a different malicious process is created that creates a child process that floods the network with illicit traffic. If the student uses the Delete tool on the infected process, a warning message declares that a system restart is required, as the system moves into Fail-Safe mode. Red lights flash and an alarm sounds. If the student invokes a restart at that point, the parent process will re-spawn the malicious child when the restart completes. To remove the parent process the student must first delete its startup file from the disc drive, but this file cannot be deleted until after the child process is deleted, which means the student must ignore the lights and alarms until after the file is deleted.
A.7.1 Learning Challenge 1: Good program behaving poorly

In Learning Challenge 1, the player is to identify a system program that is consuming too much CPU. Player is expected to correct the problem with the Nice tool to reset the priority of the program.

Initialization:

1. Spawn player on top of ring that encircles the elevator platform.
2. As soon as player steps on elevator, spawn processes and move platform down.
3. After a delay of 20 seconds, announce (audio) that system response has slowed and flash yellow.
4. Set the droid demand level of the haywire process (Exp.exe) to maximum.

Successful player actions:

1. Player uses the investigate tool on the haywire process panel or droids to verify that it is the culprit.
2. Player acquires the nice tool from its pickup location.
3. Player selects the nice tool using the scroll wheel on the mouse.
4. Player uses his nice tool on the haywire process to bring it under control.

Exceptional player actions:

1. If the player uses the "Nice" tool on a droid, the droid will slow down.
A.7.2 Learning Challenge 2: SPAM Bot

In mission B, the player is on a search-and-destroy mission for an malware program that is spewing email SPAM over the network. This mission will teach the player about the relationships between droids, where droids move, and the programs the droids represent. The SPAM Bot program (Ladybug, a.k.a. LBUG.exe) has commandeered the player’s system to send e-mail without the computer user’s knowledge. The player is expected to figure this out by noticing that the (uninfected) email process (MBOX.exe) is spewing out a lot of droids as well as receiving a lot of LBUG droids. The email program has not been infected, it is just being used a lot by the SPAM bot. So the user cannot solve the problem by ”nice”ing or destroying the email program. The player needs to go to the CPU area and see which droids are interacting with the email droids.

Initialization:  Assume that Learning Challenge 1 has completed successfully, then...

1. System places SPAM email text in a file in the ”Documents” directory on disc. This file is labeled semi-innocuously as TMP0315.

2. System spawns SPAM bot program (Ladybug), but we will likely need others for replay when user struggles to complete the mission the first time through.

3. System announces mission objectives: ”Network traffic has slowed down, please investigate.”

Successful player actions:

1. Player discovers a flood of droids associated with the uninfected email program. The player investigates and discovers that the droids spawn immediately after each of the
LBUG droids reaches the MBOX panel. Investigating the droids shows that the payloads are similar (random excerpts of a “Nigerian money order” scam).

2. Player identifies Ladybug as a ”bad” program, returns to LBUG process panel and uses the destroy tool on it to destroy the Ladybug process.

Exceptional player actions:

1. Player uses ”destroy” tool on the e-mail program panel. This does not cause a system restart, but also fails to solve the warning condition caused by a slow network.

2. Player deletes the SPAM e-mail text file.

A.7.3 Learning Challenge 3: Evil zombie program

In this challenge, a program (Fbun.exe) that has illicitly labelled itself as a required system process, has one job. On each restart, it spawns a child process, also labelled a required system process (CNt.exe) that floods the network with illicit traffic. Deleting either of these processes puts the system in FailSafe mode with the announcement that a restart is required. Performing an immediate restart does not solve the problem, as the child process is re-spawned due to the start up file (Fbun.exe) being located on disc. That file cannot be deleted while the child is alive because of a FILE-IN-USE flag. The solution is for the player to destroy the child, then delete the startup file (Fbun.exe) and then perform a restart.

Initialization:   Assume that Learning Challenge 2 has completed successfully, then...

1. System spawns child program (CNt.exe)

2. System spawns parent/source program (named Fluffy Bunny), and plants a startup file on disc
Successful player actions:

1. (To eradicate parent): Player destroys parent, then destroys child

2. (To eradicate startup): Player destroys child, then deletes file

3. (If both): Destroy parent, destroy child, delete file

Exceptional player actions:

1. If destroy child before parent, child re-spawns.

2. If try to delete startup file before destroying child, FILE-IN-USE error prevents
Appendix B

Planning Details

In addition to its tutorial functions, Annie incorporates an extensive set of graphical user interfaces for the creation and editing of operator libraries and planning problems, adapted from a system called Bowman which was also created by the author of this dissertation in the course of his graduate work. Bowman was combined with Annie partly in hope that its GUI will help encourage other researchers to develop plan-based representations of game worlds in which Annie could be tested. Annie is able to leverage Bowman’s editing capabilities, as well as its ability to select and communicate with planners, through TCP messaging to user-specified URLs or calls to referenced DLLs of planners co-resident on the same system as Annie. Several examples of Annie’s graphical renderings of plans generated for the evaluation of the FixIt pilot domain are shown in the this section.

B.1 Planning Library

An excerpt of the XML representation of the planning library can be found in Figure B.1. Readers familiar with automated planning may be surprised at the verbosity of this representa-
tion. In part this is due to the readability emphasis of XML, but it is also an artifact of library extensions enabled by Annie. In particular, extensions for variable types, and for automatically generated literals based on those types contribute to the length of the XML. Figure B.1 contains an excerpt of the XML-formatted plan returned by a planner to Annie.

![Operator Library XML Excerpt](image)

Figure B.1: Operator Library XML Excerpt
Figure B.2: Plan XML Excerpt
B.2 Pilot Plan

Figure B.2 shows Annie’s rendering of a plan space generated for the tutorial exercise used in the pilot evaluation of FixIt. Plans with the fewer flaws are shown in progressively fuller shades of yellow. Plans with zero flaws are shown in green. A user of Annie can select different plan nodes for further investigation by clicking on them.

Figure B.2 shows how a the graph for a particular plan node is rendered. Each step in the plan is depicted by a composition of rectangles. The center rectangle contains the operator name associate with the step. The smaller rectangle on the left contains the preconditions, and is labeled “P”. The smaller rectangle to the right, labeled “E” contains the effects of the action. The rectangle on top of the larger center rectangle contains the constraints. Causal links are shown in green, annotated with the associated literal, and ordering links are shown in blue.

Finally, Figure B.2, shows a more readable zoomed-in display of the center portion of that plan graph. This screen shot was taken as the moved the mouse over the little box labelled “P” on the rectangularly depicted step associated with the “FZPlayerSelectTool” action shown on the right hand edge of the image. As a result of this mouse over, the translucent window that shows detail of the preconditions appears to slide out toward the left from the box labelled “P”. As the user moves the mouse over the set of preconditions for a given step (in this case there is only one), that condition and its causal link are highlighted in yellow.
Figure B.3: Portion of Plan Space Graph from Pilot Evaluation
Figure B.4: A Plan Graph from Pilot Evaluation
Figure B.5: Plan Graph Detail
Appendix C

Main Evaluation Materials

C.1 Pre-Assessment

As described in section 5.4.1, before beginning the game session, students were given a short survey to complete. The survey is shown in Figure C.1. A pre-test knowledge score was derived based on each student’s answers to questions 6, 7, and 9. Although it was initially intended for questions 8 and 10 to contribute to the student’s pre-assessment score, the experimenter decided to exclude these due to student comments describing confusion created by poor wording of the questions. The answers to 6 and 7 were scored in a three step process. First, all short answer responses provided by all the students for each question were combined into a list for each question. Then a grading rubric was established assigning points to each answer based on its strength relative to all the other answers. Finally, the rubric was applied to each student’s survey and the points were tallied.
C.2 FixIt Adjustments

The main evaluation did not begin well. The first two students to participate triggered an identical software fault in the Unreal Engine software used by the FixIt game that caused FixIt to lock up, in each case after approximately ten minutes of play. As the students were moving their avatar through the world, it suddenly became frozen in one spot and could not be moved. Over the course of the previous year and a half of development, this error had never been observed by the developer. A slew of faults in Unreal’s handling of various events had been discovered and workarounds had been found for all of those, but this error was new. Obviously the data for both these students had to be discarded. Although the error was intermittent, the developer of FixIt was able to find a set of conditions that reproduced the error fairly often. No errors were found in FixIt code, but the developer reasoned that the cause may be related to FixIt’s need to destroy more objects of a particular type than would often be destroyed in games more typical of those built on the Unreal Engine. A workaround was devised to first relocate these objects to an area of the game world beyond the reach and view of the player. After the code was updated with this workaround, the frequency with which the intermittent error could be reproduced decreased to near zero.

However, the error was seen once or twice in further testing equivalent to perhaps one hundred evaluation sessions. Because the severity of the error was such that it would invalidate an entire session’s data collection, the experimenter reluctantly decided to add an additional safeguard of terminating the game for all students after eight minutes of play, as the error was not encountered in shorter game play sessions. Although the short time limit ensured that all students given equivalent opportunity to participate in the game, it meant that Annie had less time to prove itself with each student. Fortunately, with the software workaround and eight minute time limit in place, none of the subsequent 28 subjects in the study encountered the lock up
error or any other error that would invalidate the data collected in their evaluation session.

C.3 Post-Assessment

The post-assessment survey given to each student contained two pages: page 1 is shown in Figure C.2 and page 2 is shown in Figure C.3. Post-assessment knowledge scores were based on answers to questions 10 and 11. The remainder of Likert-scaled questions on page 1 were not expected to correlate with any of the treatment conditions.

Page 2 attempted to differentiate between the automatically generated help given by Annie (which was delivered in a “robotic” female text-to-speech voice, from the static, non-adaptive, pre-defined prompts given by a pre-recorded male voice. The answers given on page 2 were ignored for student in the control group as they did not have any help delivered in the female voice because Annie was precluded from helping them.
Circle or fill in the best answer for each question. For the purposes of this study, the term “computer malware” encompasses any form of malicious software, including viruses, worms, and spyware.

1. I understand the basics of how computer operating systems work.
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

2. I have a general understanding of what computer malware does.
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

3. I do well in computer science courses.
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

4. I am interested in learning more about how computer malware works.
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

5. I am interested in learning more about how computer operating systems work.
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

6. Name or describe three different techniques that malware can use to cause harm to a computer.
   1) ____________________
   2) ____________________
   3) ____________________

7. Name or describe three different techniques that can be used to fight computer malware.
   1) ____________________
   2) ____________________
   3) ____________________

8. For many kinds of malware infections, a system restart will probably clear things up.
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

9. I understand how computer malware can try to avoid eradication by spawning new processes.
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

10. Any malware that seriously compromises a computer’s security will noticeably affect its performance.
    a) Strongly agree
    b) Somewhat agree
    c) Undecided
    d) Somewhat disagree
    e) Strongly disagree
POST-GAME ANALYSIS

Study Participant Code #: ___________________

Circle or fill in the best answer for each question. For the purposes of this study, the term “computer malware” encompasses any form of malicious software, including viruses, worms, and spyware.

1. As I started trying to fix each problem introduced in the game, I usually had a good idea what the problem was:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

2. As I started trying to fix each problem introduced in the game, I usually knew what I needed to do to fix it:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

3. The game makes it clear what the player is supposed to do at each point:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

4. It was clear to me when what I was doing was either right or wrong:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

5. The game was frustrating at times:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

6. The game was easy:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

7. The game was fun:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

8. The game was fun when compared to other educational games:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

9. The game seemed educational:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

10. Malware can use trusted system processes to do its work:
    a) Strongly agree
    b) Somewhat agree
    c) Undecided
    d) Somewhat disagree
    e) Strongly disagree

11. Eliminating malware can require terminating processes that are not doing harm to the system:
    a) Strongly agree
    b) Somewhat agree
    c) Undecided
    d) Somewhat disagree
    e) Strongly disagree
12. I noticed that there were two different voices producing instructions and help during the game:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

13. I understood what was being said by all the voices in the game:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

For questions 14-25, if you cannot differentiate who said what, just give the same answer for both of the questions in each even/odd pair.

14. The instructions given by the deep male voice helped me know what I needed to do:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

15. The instructions given by the female robotic voice helped me know what I needed to do:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

16. The instructions given by the deep male voice helped me make faster progress in the game:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

17. The instructions given by the female robotic voice helped me make faster progress in the game:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

18. I would have liked to hear more frequent prompts similar to those given by the deep male voice:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

19. I would have liked to hear more frequent prompts similar to those given by the female robotic voice:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

20. The deep male voice seemed to know what I was doing at the time:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

21. The female robotic voice seemed to know what I was doing at the time:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

22. The prompts given by the deep male voice sometimes came too late to help me:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

23. The prompts given by the female robotic voice sometimes came too late to help me:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

24. The deep male voice helped me understand what was going on in the game:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

25. The female robotic voice helped me understand what was going on in the game:
   a) Strongly agree
   b) Somewhat agree
   c) Undecided
   d) Somewhat disagree
   e) Strongly disagree

Figure C.3: Main Evaluation PostAssessment (page 2)