Annie: Automated Generation of Adaptive Learner Guidance For Fun Serious Games

James M. Thomas, *Member, IEEE,* and R. Michael Young, *Senior Member, IEEE*

Abstract—This paper describes some of the difficulties inherent in building intelligent educational games, specifically the challenge of integrating pedagogy with core game play. We introduce a plan-based knowledge representation that provides a novel framework for infusing the core mechanic of a game with pedagogical content. We describe in detail a system that leverages this framework to dynamically adapt a game to individual learners at run-time.

Index Terms—Artificial Intelligence, Intelligent Tutoring, Plan Generation, Games, Computer-Managed Instruction.

1 Introduction

Intelligent Tutoring Systems (ITS) have advanced to the point where many research-based systems exceed the teaching effectiveness of traditional classroom-based instruction [1]. Meanwhile, commercial digital games have made enormous progress in graphics, human-computer interaction, and economic impact. Both fields include a broad diversity of applications and genres, with intriguing confluences of goals and challenges. Of particular interest to learning technologists is how learners can be guided in systems that seek to maximize user autonomy within bounded structures of pedagogy, simulation, or narrative. Although detailed studies [2], [3] have been made of learning techniques exemplified in digital games, there is little agreement about the rules or best practices that lead to effective game-based learning.

Cynics of digital game-based learning, or *serious games*, have noted that too much emphasis on learning tends to "suck the fun out" of games, to which the rejoinder is offered that too much focus on game play "sucks out the learning" [4]. Simplistic juxtapositions of teaching and game elements are notoriously disappointing. An understanding of the fundamental constructs of games, and learning within games, is needed to build systems that are intelligent, foster learning and function as games. We believe that the key challenge is to integrate learning content with what game designers describe as the "core mechanic" of each game.

This paper introduces a system we have named "Annie" that use a novel knowledge representation to integrate arbitrary sets of learning goals within the core mechanics of games. Our goal is for Annie to facilitate building games where the learning and the fun are so deeply intertwined that neither can be "sucked out". The inspiration for Annie is Anne Sullivan, who taught the blind and deaf Helen Keller how to communicate with words. Annie will need to cope with a highly uncertain model of the student's understanding while continuing to gently guide the student through trial-and-error learning typical in exploratory environments, just as Anne Sullivan coped with Helen's highly restricted communicative bandwidth and unrestrained autonomy.

2 Background

2.1 Shared Models of Guided Exploration in Games and ITS

The goals for guiding players through digital games, as articulated by game designers, are remarkably similar to the goals articulated for the guidance of ITS users. Game designers guide the player along a "golden path" [5] or "optimal game play corridor" [6] as depicted in Figure 1. Increasing challenge at any point in the game increases the risk of player frustration. If the challenge is too low, the player may become bored. Some designers stress that the sinuous shape of the player's path is as important as its position relative to the "walls" of the corridor. This serpentine route produces psychologically desirable rhythms of arousal and reflection. Game industry insiders generally credit psychologist Mihály Csíkszentmihályi's work on "Flow" [7] as inspiration for the game industry's conceptualization of guided learning. Replicas of the chart shown in Figure 1 frequently accompany discussions of learning in context of games [8], [6]. Readers familiar with learning theory may recognize the similarity between the "optimal game play corridor" and Vygotsky's "Zone of Proximal Development" or ZPD [9], [10]. Intelligent tutoring systems adapt to the individual needs of the learner to maximize the student's time in the ZPD.

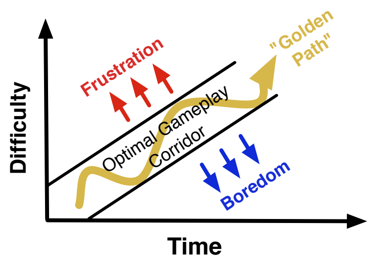


Fig. 1. The Optimal Game Play Corridor

2.2 Significant Results in Guided Exploration

This encouraging confluence between the corridor that drives optimal game design and the ZPD that guides ITS is mirrored in an extensive research record. Following the early 1980's boom in commercial gaming, Smithtown [11] and subsequent systems [12]–[14] have demonstrated successful guidance techniques for students in game-based learning environments. However, success has been more limited in exploratory, inquiry-based [15] environments. As de Jong noted [16], the research community has not yet settled on a general approach to balance guidance with student exploration "in such a way that learning is supported effectively, but the inquiry process is not reduced to following cookbook instructions."

Gee describes [2] a fairly cohesive set of learning principles used to guide players in commercial games. Game learning has advanced to the point that most players ignore game instruction manuals, expecting that the game itself will provide instruction as necessary. This expectation is the culmination of an evolutionary process fueled by the enormous market pressure within the games industry. Games that do a good job of guiding new players sell better than those that do not.

One of the guidance techniques found in nearly all games is *staggered instruction*. Modern games have become too complex for the game to try to tell players everything they need to know before play begins. 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." [2] Games often employ ITS-like hints on demand, and some games present spontaneous hints to the player. However, in contrast to the help provided by ITS systems, these hints are often not tightly linked with what the player is doing at that instant.

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 principle is check-pointing 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 state it was in when the episode began. Another way to describe this is that 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 games lack are deep models of the user's knowledge of the domain. The closest analogue to a student model is often the user's inventory of objects and skills, and the performance history of cleared checkpoints. These are usually 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 certain items in their inventory and can require those items to be used to make further progress. The extreme shallowness of these tools for modeling student knowledge belie the lack of individual adaptation in games. Instead, the game producers rely on extensive and expensive play testing to statically calibrate the potential challenges in the game to reach the widest possible audience of players.

2.3 Background Summary

Although ITS and games seem to want to do the same things, their methods are not easily reconciled. Digital games tend to favor user initiative and autonomy at the expense of fine-grained adaptation. Games are engineered to accommodate a broad range of users with the challenges and help strategies statically fixed at design time. ITS researchers try to optimize learning effectiveness and efficiency through deep user models that dynamically adjust scaffolds at run-time based on individual differences in user performance. ITS systems adapt to subtle details of learning concepts, but tend to constrain student initiative and autonomy much more than do digital games. A prime motivation in our work is to define a framework that integrates these divergent strategies in such a way as to maximize the benefits and avoid the disadvantages of each approach.

3 Foundational Perspectives on Game Play Design: A Science of Fun?

Even though play is a ubiquitous feature of human culture, it is difficult to construct a precise, yet universal definition for "game". Even more difficult is pinpointing exactly what makes a game fun. When college students are taught how to build digital games, it is common to incorporate portions of the small but vibrant literature on game and play. Although these sources do not provide a codified set of rules that could easily translate into the automated generation of "fun", they describe concepts that can be emulated.

In the promisingly-titled *Rules of Play*, Salen and Zimmerman partition their extensive exploration of games into three schemas: **rules**, **play**, and **culture**. In their view, **rules** focus on the logical or mathematical structure of the designed game system, while **play** describes the human experience of that system, and **culture** defines the larger social contexts in which the game is situated. Play is predicated by an implicit contract that if the prospective player adopts a playful or *lusory* attitude toward the game, there will be a pleasurable reward. Why is it necessary for the player to agree to adopt a particular mental attitude to play a game? Salen and Zimmerman refer to an insightful description of the game of golf:

> Suppose I make it my purpose to get a small round object into a hole in the ground as efficiently as possible. Placing it in the hole with my hand would be a natural means to adopt. But surely I would not take a stick with a piece of metal on one end of it, walk three or four hundred yards away from the hole, and then attempt to propel the ball into the hole with the stick. [17]

Thus, a successful game demands the player acquiesce, to some extent, to endure arbitrary impediments to the goal. "In anything but a game the gratuitous introduction of unnecessary obstacles to the achievement of an end is regarded as a decidedly irrational thing to do, whereas in games it appears to be an absolutely essential thing to do." [17] The player's tenuous acceptance of the rules of the game is why game designers place such high value on "immersion", and maintain vigilant focus on the pacing and delivery of their part of the bargain: fun.

Salen and Zimmerman state that goal of successful game design is to create what they describe as "meaningful play", which they claim "emerges from the relationship between player action and system outcome; it is the process by which a player takes action within the designed system of a game and the system responds to the action. The meaning of an action resides in the relationship between action and outcome." [18] Koster echoes this, saying that a key component of successful games is a solid core mechanic, which he defines as "an intrinsically interesting rule set into which content can be poured" [19]. Adams and Rollings describe the core mechanic of games as "one or more causally linked series of challenges in a simulated environment" [20]. The common thread in each of these academic characterizations is that the core of good games is embedded in the rules of action, cause and effect.

Our work extends a well-understood computational model of action, cause, and effect both to drive the generation of game play elements and to represent the state of the student's knowledge of the game world. This model is built using the language of automated planning, specifically the STRIPS-style [21] descriptions of actions, effects, and objects in a planning domain. By using the same model for both the game play elements in the world, and our model of the student's understanding of that world, Annie is centered on what Salen and Zimmerman describe as "meaningful play", which they claim arises from a tight coupling of action and outcome.

4 A Plan-Based Representation of the Core Mechanics of Digital Game Play

The style of action descriptions invented for the STRIPS [21] system in the early 1970's have continued to form the basis of much subsequent research in automated planning . Building on several distinct approaches to integrating automated planning with game domains [13], [22]–[24], we introduce a general plan-based knowledge representation that is intended to be applicable to any game domain.

4.1 Operator Descriptions

Figure 2 depicts a STRIPS-style operator, a schematic description of a family of operators (or *tasks*) in a standard planning representation. The name of this operator is **deleteFile**. It has two parameters: the initiator of the task and a fileDescriptor naming the file to be deleted. Planning researchers use the convention of reserving the word "action" to describe a task or operator that has instantiated with particular set values for the initiator and the fileDescriptor. Thus, a task description is a schematic representation of possible varieties of **deleteFile** actions that could arise from different variable bindings.

To the left side of the rectangle in Figure 2 are shown the two preconditions that must be satisfied before **deleteFile** can execute, and shown to the right its post-condition or "effect". Together, these conditions say a file will no longer exist following the execution of deleteFile, but the file must first exist and not be inUse before the deleteFile action can execute. The final condition associated with **deleteFile** is the constraint shown above the rectangle that that says the object bound to ?initiator must be the student or player of the game. Constraints are special cases of preconditions that can be evaluated at initialization time because their truth valuations will remain invariant over the course of execution of the plan (e.g., variable types like "isStudent"). Although space restrictions prevent us from depicting a more complex example, this representation also supports hierarchical tasks, that is, tasks composed of sub-tasks. To build the model for

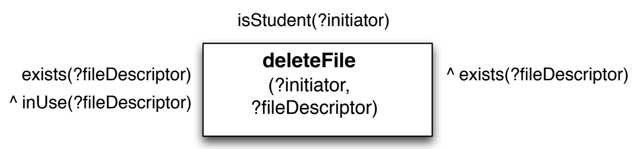


Fig. 2. Example Task Description

what the student knows about deleting files, Annie begins by automatically deriving a set of meta-conditions from the known features of the **deleteFile** operator. For example, Annie can generate the following set of conditions to capture the state of the student's knowledge of the deleteFile operator at any point in time:

- hasConstraint(**deleteFile**, isStudent(?initiator))
- hasPrecondition(**deleteFile**, exists(?fileDescriptor))
- hasPrecondition(**deleteFile**, ¬ inUse(?fileDescriptor))
- hasEffect(**deleteFile**,¬ exists(?fileDescriptor))

This is precisely the basis for Annie's model of each student's knowledge of all the operators in the game world.

4.2 Student Modeling

The simplest model of the student's knowledge of the operators in the domain would register whether the student "knows about X" or "doesn't know about X", where X is one of the meta-conditions that describes the operators in the world. This simple model fails to capture the perennially uncertain nature of student knowledge in an exploratory environment where the student's understanding of the world evolves *gradually*. To represent this uncertainty we have chosen a rough-grained five-valued scale (**HighlyLikely, Likely, Neutral, Unlikely, HighlyUnlikely**) to represent varying estimates of the likelihood that the student believes or knows about a particular facet of the domain, where "Neutral" is the default initial value. Future work may investigate the usefulness of a more sophisticated student modeling based on traditional A.I. belief systems based on modal logic. By comparison the current representation is impoverished in that is does not allow for reasoning of "knowing about knowing X".

To illustrate, in a game that teaches the processes involved in aerobic cellular respiration, Annie may observe a student behavior that implies that the student knows an effect of the Krebs cycle is the production of CO_2 waste but may have no information yet on whether the student knows another effect of the process is the production of H_2O. This could be represented in the student model by marking the **hasEffect** condition corresponding to CO_2 production of a particular action in the Krebs cycle as **HighlyLikely**, while the effect that produces H_2O is marked as a student belief with **Neutral** likelihood.

This representation may seem unnecessarily complex. The rules for handling *deleteFile* could be specified much more simply. However, our focus is not the individual operators, but rather their application in the larger structures of individual plans, and more usefully, the space of all possible plans that delivers value to Annie in the form of valuable inferences. To that end, we will skip over some of the high-level description of Annie to investigate the possible benefits of plan space exploration for serious games.

4.3 Plan Space Exploration

A potentially difficult paradox for us is that 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 inferences. In order to characterize how close a student is to achieving important goals or milestones within the game world, we compute the game world's *plan space* – a directed graph that characterizes the space of all possible plans for a given planning problem. In this graph, nodes represent (possibly partial) plans and an arc from one node to the next indicates that the second node is refinement or more complete version of the first. Planning algorithms called *plan-space planners* [25] construct plan spaces as part of their search process when solving a planning problem. Each plan in the plan space differs, in part, due to the variations in the sequences of actions they use to achieve their goals. Because the proper sequencing of actions within a plan relies on valid student knowledge regarding the tasks involved, Annie can use the plan space to prioritize and sequence its strategies for guiding the student toward acquiring the requisite knowledge. Annie does not introduce any new deductive reasoning to make sense of the actions in plans, but merely leverages the power of plans and plan space representations provided by traditional planning.

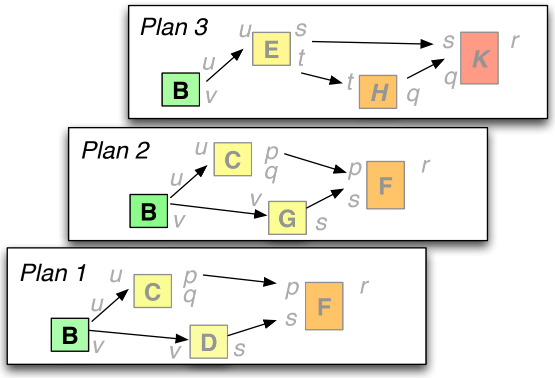


Fig. 3. Alternative Plans From the Plan Space

For example, Figure 3 depicts a situation where action B has completed successfully. The upper case letters represent actions and lower case letters in the Figure represent preconditions and effects. The most proximal actions to consider are those shaded yellow {C, D, E, and G}. Any of those actions could be executed next, as all of their preconditions are currently satisfied, i.e., zero intermediate steps are required before they can execute. This group of actions can be called "Tier 0". The next most proximal tier of actions (Tier 1 = {H, F}: shaded orange) are those whose preconditions can be satisfied by a causal chain of length one. Finally, action K is in tier 2, because it cannot execute until both of its preconditions are satisfied, and the shortest chain that establishes precondition q is of length 2. Thus, we have a numerical method (tier numbers) to rank goal proximity across the plan space. Annie can use this to diagnose student misconceptions and generate timely guidance.

Annie is designed to exemplify the strengths of a human tutor in being extremely attentive to the actions of the student, deferential to student initiative, but skillful and clever when intervention is warranted. These are ambitious goals. The plan space gives Annie a foundation for the generation of intelligent tutoring behaviors in the context of games.

5 Integrating Game Mechanics With Learning

We have argued that the most successful educational games tightly integrate the pedagogical elements with game play. We have shown that the divergent implementation methodologies of ITS and games preclude a facile synthesis, or "bolting-on" of one with the other. We have presented one model for a knowledge representation that could help bind educational content with game play. The goal is the dynamic generation of game play elements that guide a player experience that is both educational and *fun*. What we have yet to describe are the prevalent approaches to *fun* in digital game design.

5.1 Conventional Game Design

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 less than three years, 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 [26].

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.” [26] 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 as the *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.” [27] 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 game play, and helps disguise the essentially static structure of the underlying *spine*.

The static structure of digital games is an old story. Crawford stated flatly in the mid-1980’s that the structures underlying digital games are no more than ”branching tree structures.” [28] 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 dwarf those of the early era, it is still the case that almost all games are no more complex than branching tree structures. 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.

We believe that a plan-based representation can provide a language to describe simultaneously learning content and game play. With automated planning techniques, we can ensure that the *spine* of the game is traversed, while encouraging the player to explore far beyond the small set of detours built into a *golden path*. Through planning, a widely varied *golden landscape* unfolds where individual users can explore a wide variety of experiences tailored to their particular educational and entertainment aspirations.

The Mimesis [24] and Zócalo [29] systems, upon which Annie is based, showed that automated planning can be combined with industry-leading game engines to deliver dynamically adaptive content. This foundation provides Annie with the fine-grained control required for ITS-style dynamic adaptation. As will be shown in the following section, Annie’s contribution is a framework that leverages the teaching principles and conventions proven to be effective within games.

5.2 Recapitulating Game-Based Learning Through Planning

The knowledge representation described in Section 4 is not meant to be definitive. Other plan-based models could be constructed, including those based on different theories of learning. Our immediate interest is not in providing a perfect model, but rather in proving that a model of this general type can be used to drive the dynamic generation and adaptation of learning principles in games.

Gee described a rich set of learning principles evident in commercial games [2] and Quintana described a framework that described many of the scaffolding techniques used in exploratory ITS research [15], but neither of these descriptions lends itself to a generative model. Each leaves it to the artistic spirit of game or tutorial designers to decide when, where, and how extensive the computational support should be.

Annie, however, requires a generative model for game-based learner guidance. We have built such a model inspired by the descriptions of Gee and Quintana. To the extent that this generative model is successful, it will provide the following capabilities:

1) **Synthetic Generation.** Each learning principle is articulated through one or more plan-based templates to allow automatic generation of game play elements that embody that principle, rather than expensive and time-consuming human-authored design elements.
2) **Dynamic Adaptation.** A secondary advantage of synthetic generation is that it allows for generation to be performed at run-time, where the game can dynamically adapt to the behaviors exhibited by the student.
3) **Quantifiable Extent.** With the ability to generate and adapt as needed comes the capability to measure or specify the frequency and extent to which learning principles are realized. In other words, it provides researchers with a mechanism to freely vary the prevalence of one principle vs. another and measure the effects.

A detailed examination of the 36 learning principles articulated by Gee revealed a set of nine of these principles that appear to be good initial candidates for testing this generative model. Three of these are described in detail in the following subsections. Space limitations prevent us from describing all nine in detail. Each description begins with Gee’s characterization of the principle, followed by our identification of the quantifiable properties we derive from the description, and a high-level sketch of generative, plan-based computational patterns that could be used to dynamically instantiate the principle in a game.

In Section 6, we describe the core algorithm and provide a worked example showing how these principles are used to dynamically construct plan-based learning experiences.

5.2.1 Discovery Principle

“Overt telling is kept to a well-thought-out minimum, allowing ample opportunities for the learner to experiment and make discoveries.”

Quantifiable Properties. We use the term *remediation* to describe an action Annie inserts into the game environment to attempt to correct what it perceives to be a misapprehension on the part of the student. We can count the number of remediations applied for each student, the best-case, worst-case and average number of remediations required for each particular knowledge component, and the comparative frequency of stronger or weaker hints that correspond to different type of remediations. Across a broad range of students, these measurements can be used to characterize the difficulty of different parts of the game world and help pinpoint areas where more student guidance opportunities may be required.

Computational Pattern. Remediations are organized in such a way as to allow software to choose between successively more explicit modes of instruction. This builds on extensive ITS research into the optimal selection strategy between the frequently used guidance options of ‘Prompt’, ‘Hint’, ‘Teach’, or ‘Do’.

5.2.2 Multiple Routes Principle

“There are multiple ways to make progress or move ahead. This allows learners to make choices, rely on their own strengths and styles of learning and problem-solving, while also exploring alternative styles.”

Quantifiable Properties. Annie can quantify the number of distinct successful plans, the number of qualitatively different plans in the plan space, the number of actions that must be included in any successful plan, or even the ratio of the number of these critical actions to the mean total number of actions in successful plans.

Computational Pattern. Annie allows for extensive mining of the space of potential plans to reveal bottlenecks, potential for off-task activity, etc. in a way that could be much cheaper and more extensive than traditional game design play testing strategies.

5.2.3 Explicit Information On-Demand and Just-in-Time Principle

“The learner is given explicit information both on-demand and just-in-time, when the learner needs it or just at the point where the information can best be understood and used in practice.”

Quantifiable Properties. The timeliness of explicit information can be measured by the duration between when the information is provided and when it is needed. This can be compared and contrasted with the number of opportunities for on-demand information in the environment. For some students or groups of students Annie may want to vary how far in advance help can be provided based on projected memory persistence of those students. As post-hoc measurements, analysis of these properties over many students can be used to calibrate the guidance within Annie.

Computational Pattern. As described in later sections on system implementation, the Annie system continuously calculates the immediacy of information requirements in terms of proximity of plan operators in successful plans.

5.2.4 The Other Six Principles

Space does not allow for a full description of the remaining six of Gee’s principles that Annie currently implements. However, they are briefly listed and described below:

1) **Incremental Principle.** Order challenges so that complex situations build on earlier, simpler examples.
2) **"Regime of Competence" Principle.** Provide only as much support as the student needs to avoid frustration.
3) **Semiotic Principle.** Use a broad range of sign systems to communicate pedagogical content.
4) **Achievement Principle.** Provide intrinsic rewards customized to individual performance to signal mastery.
5) **Practice Principle.** Provide paths to success that allow for repetition and even failure.
6) **Transfer Principle.** Vary the levels of specificity and generality in learning content.

5.3 Summary: Advantages of Plan-Based Game Design

We have based our model not on a scientific theory of learning, but simply a survey of learning conventions evident in commercial games. Clearly, a knowledge representation based on a more powerful theory of learning may reveal profound advantages. Our immediate intention is simply to demonstrate that a nominal plan-based knowledge representation can lead to a computational framework that can automatically synthesize and adapt game/teaching at an atomic level. If this demonstration is successful, it should motivate future work to discover useful revisions to the knowledge representation.

We have selected a set of learning principles derived by a scholar of digital games and shown how a plan-based design can enable the realization of these principles in arbitrary domains. Specifically, our proposed knowledge representation synthetically generates game structures that implement these principles, requiring less time, and money, resulting in a shorter and cheaper development cycle. Because these structures are automatically generated, their instantiation can be shifted to run-time, so they can be tailored to the immediate and subtle learning needs of the individual rather than the statically defined and obvious extremes of an entire population. Finally, the rules governing how and when to change course are visible and modifiable, rather than deeply entwined with tutorial algorithms. This enables the system to conform to externally-specified metrics for particular applications.

The use of a plan-based knowledge representation breaks the game spine into interchangeable parts, allowing for dynamic synthesis of game progression while ensuring that the player eventually traverses segments of the spine nominated as particularly critical. Any fixed branching structure could be implemented through a plan-based representation by representing each critical action choice as a distinct operator with unique prerequisites and effects. But planning not only replicates the expressivity of existing game progression, but allows for a much wider variety of scaffolding techniques, partial-ordering of actions, and varied bindings of particular game elements (e.g., any organic molecule as opposed to particular phosphate molecule instance 314), and arbitrary number of repetitions or cycling through particular types of actions.

6 Implementation Overview and Example Application: Annie and FixIt

6.1 System Implementation: Annie

Annie is an extension of a system, called Zócalo [29], that generates interactive narrative for 3D game worlds through a planning-based knowledge representation. Annie uses plan construction techniques to build and dynamically adapt tutorial plans that guide learners' actions in a virtual environment. This adaptation takes the form of automatic generation and fading of scaffolding tailored to the immediate knowledge requirements of the student. All of this relies on a knowledge representation whose base element is the description of in-game tasks.

Although this burdens application developers using Annie with the responsibility of providing a plan-based description of the relevant features of the domain, modern GUI-based tools like Bowman [30] and Wide-Ruled [31] can help reduce the effort of authoring and validating these descriptions. The trade-off is that a one-time investment in providing a plan-based description is all that is required for Annie to model student progress and automatically provide support tailored to the specific learning needs of a student for any learning challenge in the domain.

Annie uses this representation to define the initial and goal states of student knowledge and provide proactive mediation within the exploratory environment to guide the student toward successful learning.

Annie constructs an initial tutorial plan consisting of a plausible partially-ordered sequence of student and system-initiated actions that is designed to bring about a specific goal state for the world, including a particular state of task knowledge acquisition in the student model. The plan marks out the optimal game play path for the user prior to the start of the session, but it is continually revised based on student actions.

Annie's execution loop iterates each time an action is taken in the world, either by the student or the system. Following the action, Annie consults an extensive library of general **diagnostic** *templates* to update its student model. A template is a collection of domain-independent plan reasoning rules that Annie uses to diagnose and remediate student misconceptions. An example of a diagnostic template is the set of rules that define a case where a student-initiated action fails due to an unsatisfied precondition. In its initial implementation, the rules comprising each template are hard-coded, but once our experimental evaluations help us determine an optimal initial set of rules, we intend to develop a schema to allow these templates to be defined external to Annie and loaded at run time. These templates are described in more detail in [32].

As shown in Figure 4, Annie is an extension of the Zócalo [29] and Mimesis [24] systems that generate interactive narrative through planning-based knowledge representations.

6.2 Example Application: FixIt

Because Annie specializes in task-based learning, it is optimally suited for domains where the key learning challenges require the student to properly compose a sequence of causally-related actions. For our initial evaluation of Annie's effectiveness we have chosen the domain of computer security viewed from the perspective of a generic operating system. This domain is the basis for a game called "FixIt" we have developed on the Unreal Tournament 3 engine. The game features four progressively more difficult missions that require the student/player to identify and remove increasingly complex forms of computer malware. The learning goal is for the user to gain a deep understanding of the mechanisms by which computers can become infected and the procedures security software uses to disinfect operating systems.

The game is constructed as a series of missions. The player's first mission is to identify a system program (shown in Figure 5) that is consuming too many resources. The player is asked to use the "Information" tool to determine the wayward process, which is depicted in Figure 5 as the process Syswatch.exe. The player is then supposed to correct the problem with the "Nice" tool to reset the priority of the process. Subsequent missions presented to the player involve malicious attacks on the computer. The challenges build on each other so that the student must reuse skills acquired in earlier missions. This helps reinforce learning while extending the relevance of the student model, providing more flexible scaffolding and fading than would be possible if the tasks were more discrete. This reuse also allows us to confront the student with fairly complex challenges.

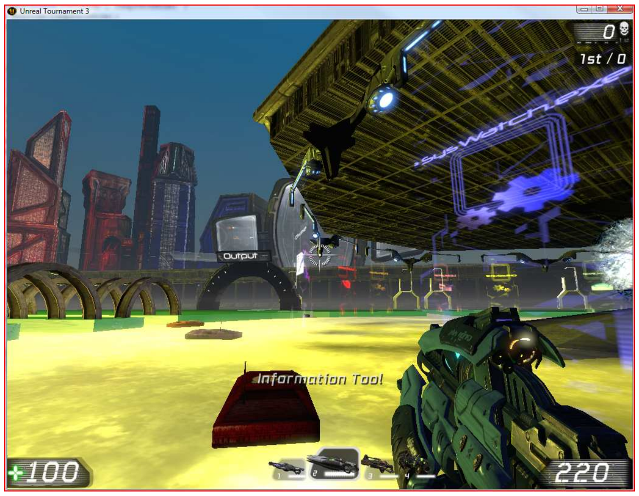


Fig. 5. FixIt: Information Tool with SysWatch.exe

For example, in the third mission a renegade child process is automatically re-spawned by a trojan hidden inside a trusted

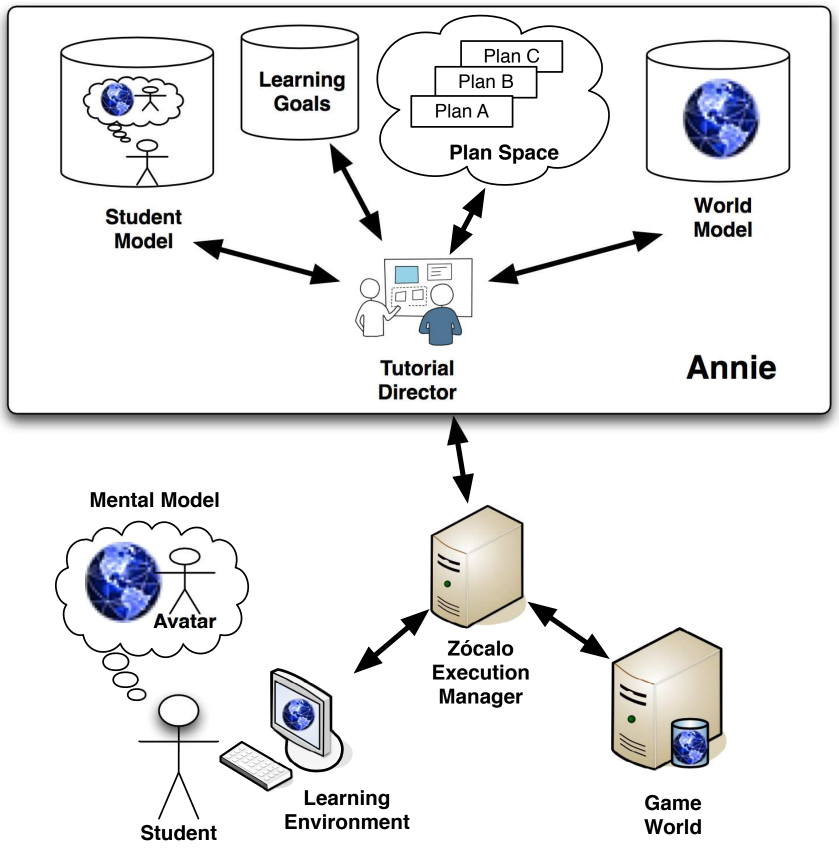


Fig. 4. System Architecture

system program. If the parent is found and killed, it triggers an automatic system restart. But the attack has also left a back door, so if the user does not close that vulnerability, the parent process will be reinfected and re-spawn the child process. The back door is a hidden file that can only be deleted when not protected by a FILE_IN_USE flag held by the renegade child. Thus, the third mission forces the user to evaluate earlier learning: that killing a bad process cures a malware infection, and shows that this is not always true. It requires that the student revise this over-generalization. Furthermore, it helps show a core domain concept: that malware is not like a stain or infection that can be cured by straight-forward excision, but it is often a multi-pronged invader that requires a multi-stage solution. By forcing the student to revisit the same tasks through a series of “missions”, where the same task is operating on different objects in the world, we are able to evaluate the usefulness of our task-based student model.

6.3 Required System Inputs

Annie is a data-driven system. External libraries of domain-independent plan-based diagnostic and remediation templates encode all the particulars of Annie’s scaffolding. At initialization, these libraries are applied to the session-specific Learning Problem Description (*LPD*), which defines a set of domain-specific learning goals for Annie to achieve in that particular session. The first component of the *LPD* is a typical planning problem description which describes the plan operators, objects, conditions, initial and goal states of the game world, without reference to any learning states or goals. This problem description is written in a restricted form of the STRIPS-like language used for all interactive narratives in Zócalo.[1]

It would be possible to embed pedagogical goals into particular operators and states of objects in the world to create a tutorial that does not exercise any of Annie’s student modeling. Indeed, most commercial games implement teaching by creating actions for the student to perform that signify to the system that the student knows some particular thing. However, one of the prime motivations for Annie is that it is often not enough for the student to simply “do” some action. It is possible for a student to perform an action without fully understanding it, either because the action was a “lucky guess”, or the student was distracted by other game elements when the action was performed. For these situations, the second component of the *LPD* allows the author to specify learning goals as a set of explicit meta-knowledge of the operators in the problem description, using predicates derived from the planning problem description, such as “hasPrecondition(*operator*, condition)” and “hasEffect(*operator, condition*)”.

1. The plan-based interactive narrative representation used by Zócalo is described more fully in [24].

6.3.1 Remediation Templates

Assume that Annie has selected the following knowledge gap for remediation:

$$hasPrecondition(\mathbf{deleteFile}, \neg inUse(?fileDescriptor))$$

This means that Annie wants to increase the student's strength of belief that the **delete-file** operator has a precondition that the file cannot be *inUse*. Annie chooses a remediation template from the library to apply to the plan. The following example shows one possible instantiation of a **demonstrate** remediation template:

1) $show(inUse(file1, app1))$. Show that file1 is in use by an application app1.
2) $deleteFile(Ann, file1)$, A system-controlled character (named Ann) attempts to delete file file1.
3) $closeApplication(Ann, app1)$, where the application app1 is the current user of file1.
4) $show(\neg inUse(file1, app1)$. Show that file1 is not in use by application app1.
5) $deleteFile(Ann, file1)$. Ann deletes file file1.

This fairly complex example was chosen to highlight that the remediation templates are not simple atomic activations, but partially-specified plan structures that Annie must dynamically weave into the particular state of the particular session to achieve defined pedagogical goals. The template is "partially-specified" in that it contains placeholders that allow Annie to find the right combination of operators and ground terms to bring about the intended changes.

6.3.2 System Initialization

At initialization, Annie uses the external template libraries, the problem description, and the rest of the *LPD* to compile its run-time knowledge base. The first task is to initialize the knowledge base with the problem description amended with pedagogically-focused elaborations of each operator. Annie uses these elaborations to represent a student's state of belief for each precondition and effect of each operator in the planning problem description. Included in this construction phase is the compilation of all the remedial and diagnostic templates to represent beliefs specific to the individual operators of the domain.

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 then uses heuristics derived from its knowledge base to choose the initial plan from the set of successful plans. These heuristics will consider the student's specified proficiency, plan adaptability, as well as traditional measures of plan structure. The planner generates a set of successful plans as well as a wider space of possible plans, both complete and incomplete. Annie uses this plan to initiate the tutorial session within the game world. Because the planner is only concerned with the states of the world, the tutorial plan does not *guarantee* that the goal beliefs for the student will be achieved. Rather, Annie monitors student behavior and optimizes the frequency and extent of its tutorial interventions to increase the likelihood that the goal beliefs are acquired.

6.4 Execution Cycle Overview

Annie's run-time behavior consists of a loop that iterates each time the student or the system executes an action in the world. As VanLehn observed [1], many seemingly dissimilar tutoring systems employ cyclic execution models. In fact, most of these can be characterized as a pair of nested loops, where the outer loop iterates over "tasks" and the inner loop iterates over each *step* in a task. Because Annie's plans contain hierarchical plan structure, its single loop sometimes iterates over tasks and sometimes over steps in a task. As depicted in Figure 6, Annie's execution cycle is divided into five *stages*. It may be helpful to think of these in the context of a cycle, since an action is actually executed in the fourth stage, and the fifth stage is where Annie updates its student model based on the action taken. When the cycle returns to the first stage, it means Annie must consider whether or not to alter the plan to remediate some student misconception based on the data analyzed in the fifth stage of the previous cycle.

6.5 Stage 1 - Remediation Consideration

First, Annie reviews the student model for unrealized pedagogical goals. It then compares the beliefs about upcoming tasks in the tutorial plan space to identify knowledge gaps that may hinder the student's immediate progress. Urgency thresholds for repairing these knowledge gaps are derived based on the proportion of steps remaining in the possible successful plans. If the threshold is met, the most critical gap is chosen for remediation in Stage 2. Otherwise, if no candidate remediation is deemed sufficiently urgent, the plan is left unchanged and Annie proceeds directly to Stage 3. The key part to Stage 1 is that Annie continually examines the current *fringe*, the proximal set of actions, some subset of which is hopefully in the mind of the student. Annie does not need to predict all of the actions the student will take in the future, but merely needs to help the student get past the immediate challenges of the next few actions in the plan and learn the beliefs specified in the *LPD*.

6.5.1 *Remediation Consideration: Calculate the* Mean Goal Proximity Ratio (MGPR)*.*

The *MGPR* provides a rough estimate of how close the student is to completing any of the remaining potentially successful plans. It is calculated as the mean of the *GPR (Goal Proximity Ratio)* of all such plans. For a given plan, the *GPR* is given by:

$$GPR = \frac{number\ of\ actions\ until\ goal}{total\ number\ of\ actions\ in\ successful\ plan}$$

Annie uses the *MGPR* as an estimate for the remaining number of useful tasks consistent with successful plans. This helps determine the urgency with which Annie guides a student toward tasks on known paths to success.

6.5.2 *Remediation Consideration: Identify and Rank the* LPD Gap Set

The *LPD Gap Set* is the subset of the *LPD* beliefs that are not currently satisfied. Annie ranks each gap according to the distance between its current belief strength and its target belief strength given in the *LPD*.

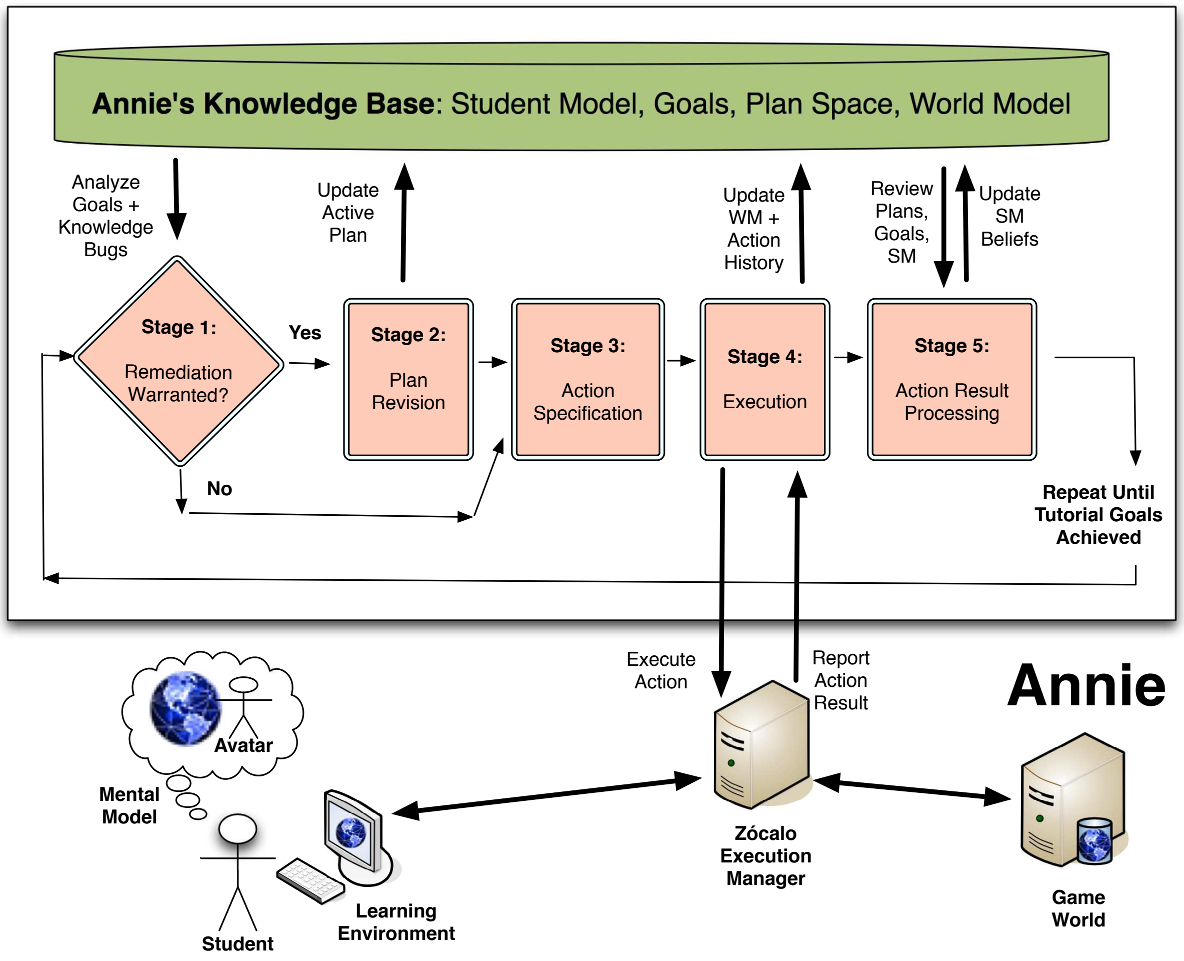

Annie's Knowledge Base: Student Model, Goals, Plan Space, World Model
Analyze Goals + Knowledge Bugs
Update Active Plan
Update WM + Action History
Review Plans, Goals, SM
Update SM Beliefs
Stage 1: Remediation Warranted?
Yes
No
Stage 2: Plan Revision
Stage 3: Action Specification
Stage 4: Execution
Stage 5: Action Result Processing
Repeat Until Tutorial Goals Achieved
Execute Action
Report Action Result
Annie
Avatar
Mental Model
Student
Learning Environment
Zócalo Execution Manager
Game World

Fig. 6. Execution Data Flows

6.5.3 *Remediation Consideration: Identify and Rank the* Proximal Belief Gap Set

The *Proximal Belief Gap Set (PBGS)* is the subset of the beliefs about the proximal actions in the set of possibly successful plans. These are ranked according to belief distance, just as with the *LPD gap set*. In addition, each belief is ranked according to the proximity of the action that contains the belief.

6.5.4 *Remediation Consideration: Select a gap that exceeds threshold.*

Empirically derived thresholds are applied to the beliefs in each of the gap sets to decide which beliefs, if any, should be remediated in this execution cycle. If none of these exceed the thresholds, Annie considers the *LPD gap set*. These thresholds will be calibrated so that Annie's proclivity for remediation is low. In most cases, no candidate remediation will exceed the urgency threshold, and Annie will proceed directly to Stage 3. Only when a gap is chosen for remediation will Annie proceed to Stage 2 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 gaps in actions that are not even required for the student's plan to succeed. This could result in Annie inefficiently oscillating between several candidate plans forcing the student to adjust to an incoherent sequence of Annie's actions.

6.6 Stage 2 - Plan Revision

The purpose of Stage 2 is to identify the plan alteration required to remedy the knowledge gap identified in Stage 1. This is accomplished in four steps. First, Annie selects a remediation. Second, Annie generates a sub-plan tailored at run-time to the current plan context. Third, the existing tutorial plan is repaired to include this new sub-plan. Finally, the repaired plan is compared to alternative successful plans in the plan space to see if one of these offers a better fit for the student's recent plan history.

At the end of the stage, if the student's recent actions show a pattern of increasing divergence from the current plan, coupled with increasing similarity to an alternative plan, Annie defers to the student's initiative and switches to the plan that seems to better match what the student is doing. Otherwise, if a remedy 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. Each of these steps is described in detail below.

6.6.1 Plan Revision: Remediation Selection

The primary focus of the plan revision stage is for Annie to adapt tutorial plans to the needs of the student by selecting from the remedial operators whose *knowledge outcomes* address the knowledge gap selected in the previous stage. The simplest variety of remedial operators correspond to common ITS interventions including **prompt**, **hint**, **demonstrate**, **teach** or **do**. In addition, Annie can be provided with hand-authored composite remedial operators particularly attuned to common misconceptions for a particular domain. If more than one remedial operator is found, Annie considers the *knowledge outcomes* and *prerequisite knowledge* that annotate each alternative, comparing these to the current state of knowledge represented by the student model and the selected knowledge gap. In addition, Annie considers the student's proficiency and progress history to determine how aggressively the knowledge gap should be remediated. Our initial work uses a simplified selection algorithm similar to the "Fixed Strategy" studied by Murray and vanLehn [33].

6.6.2 Plan Revision: Sub-plan Formation

Sub-plan formation is simply the instantiation of the selected remedial operator (likely an abstract, or hierarchical operator) with details from the current plan context.

6.6.3 Plan Revision: Sub-plan Integration

Once the new remediation sub-plan has been created, it must be integrated into the current plan. Our work extends the DPOCL planning algorithm [34] at the heart of Zócalo with a new function capable of wedging a new sub-plan into an existing plan at a particular point (i.e., so that the sub-plan is ordered to precede a particular action in the original plan).

6.6.4 Plan Revision: Alternative Plan Consideration

In the final step, Annie considers abandoning the currently active and newly remediated 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 adopt the alternative plan, rather than to continue to steer the student back onto a plan that the student has chosen not to follow.

6.6.5 Plan Revision: Example

To show how Annie would traverse the four steps of the plan revision stage for a typical case, we return to the example of the operator named **deleteFile** depicted in Figure 2 in section 4.1. We assume that the current plan calls for the student to perform a $closeApplication(app0)$ and then $deleteFile(file0)$, and that Annie has identified the knowledge gap:

$$hasPrecondition(\mathbf{deleteFile}, \neg inUse(?fileDescriptor))$$

This means that Annie is concerned that the student does not know that **delete-file** operator has a precondition that the file cannot be *inUse*. In the first step, Annie consults the set of remedial operators defined for "Unknown Precondition". Based on the current state of student learning, Annie might choose the **demonstrate** remediation.

In step two, Annie instantiates the remedial operator so that it forms the following sub-plan:

1) $reveal(inUse(file1))$. "Show" that file1 is inUse.
2) $show(Alice, opr1)$. "Show" a system-controlled character (Alice) attempt and fail to $deleteFile$ on file1.
3) $show(Alice, closeApplication(app1))$, where the app1 is the current user of file1.
4) $show(Alice, deleteFile(file1))$. "Show" a character successfully deleting file1.

In step three, this sub-plan is merged into the current plan, such that the sub-plan will complete prior to the action already in the plan that required the student to $closeApplication(app0)$. In step four, Annie may or may not choose to abandon the current plan in favor of an alternative.

This example also illustrates some of the challenges in writing sufficiently general operator templates that produce predictable pedagogical impacts. Clearly, some cleverness is required on the part of the template writer. As Annie is applied to different domains, we expect to discover some best practices for the confluence of knowledge representation and template construction.

6.7 Stage 3 - Action Specification

Annie's job in Stage 3 is to decide which of the actions in the (possibly updated) current plan should execute next. Annie remembers whether a remediation was selected in Stage 1 and if the plan was updated in Stage 2. If a remediation was selected and the plan was updated, Annie will narrow the focus to the added sub-plan. In any case, there may still be more than one potential action that can be executed next (which we have been referring to as *proximal actions*), and these actions may be either student or system-initiated.

If Annie chooses a student-initiated action to execute next, its subsequent actions may seem counter-intuitive. Because Annie cannot force the student to take a particular action, the Zócalo execution manager must process this as a null request and do nothing, no matter which particular action is chosen. Therefore, if all the proximal actions are student-initiated, Annie should abbreviate its deliberations in Stage 3, as the result will be the same no matter which action is chosen. In fact, because Annie should be deferential to student initiative, it will often be the case that a student-initiated "null" action will be chosen even when there are possible system-initiated actions that could change the state of the world in a way Annie would judge to be desirable.

In cases where the plan has been updated with a remedial action and all the elements of that remedial action have not yet executed, Annie chooses from the proximal actions of that remediation. Otherwise, Annie chooses non-deterministically from the set of proximal actions, giving higher priority to student-initiated actions.

6.8 Stage 4 - Execution

First, Annie checks its message buffer to see if it has been notified of any student actions since the last iteration. If not, Annie sends a message to the execution manager telling it the next action to be executed. The execution manager acknowledges this message and the loop continues. It might seem reasonable that in the case of student-initiated actions, Annie should wait for an acknowledgement that the student has in fact taken the action. Instead, we allow those messages to arrive asynchronously and buffer them. If, at the beginning of this stage, such a message is in the buffer, Annie discards the action that had been selected and simply carries the student-initiated action result message into Stage 5.

If the action chosen in Stage 3 is a system-initiated action, Annie send a message to the Zócalo execution manager to execute that action. The execution manager will then notify Annie when the action has completed, and the execution loop will continue. If the action chosen in Stage 3 is a student-initiated action, Annie sends a *continue* message to the Zócalo execution manager and initiates a wait loop of approximately one second that can be interrupted only by notification of student action. It is reasonable to suspend Annie's execution cycle at this point because nothing will change in the game world until the student takes an action. This ensures that Annie does not needlessly consider its entire inventory of system-initiated actions or oscillate between alternatives in a way that confuses the student.

6.9 Stage 5 - Action Result Processing

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. Annie updates its world model with the effects of that action. Annie updates its student model by finding the best match between the action result within a pre-computed catalogue of action result scenarios.

Each action attempted by the student presents an opportunity for Annie to update its model of the student's beliefs. Annie bases this reasoning 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 value in making this assumption is overwhelmingly positive.

For Annie, student model maintenance is more complex than one might assume. If a student's action fails, it should be the case that the student has at least one misconception in their mental model, but it may also be the case that this action attempt presents evidence in support of one or more 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. In any of these cases, a given belief in the student's mental model may or may not already be represented in Annie's student model. Therefore, Annie examines evidence to strengthen or weaken beliefs in the student model for both failed and successful actions.

6.9.1 Diagnostic Templates: Scenario Descriptions

Each of the diagnostic templates in Annie's library corresponds to a particular *scenario* that describes the relationship between the most recently executed actions, the plan space, and the rest of Annie's knowledge base. We describe each of the scenarios below. For one of the scenarios we provide a detailed illustration.

- **Failed Action** When an action that the student attempts fails, there is a good probability that an underlying misconception can be identified, because Annie can identify at least one precondition that was not established at the time the student's action was performed. This could be caused by a precondition that has never become true since the beginning of the session or, as shown in Figure 7 where an student's attempt to execute action C fails because precondition p, which was established by action Action A, but subsequently negated as an effect of action B. Table 1 provides a brief logical description of five possible misconceptions Annie evaluates in this scenario.
- **Successful, but *fatally flawed* action.** The action may be predicted to 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).
- **Successful, but *path-limiting* action.** A successful student action eliminates some but not all possible successful plans from the plan space (e.g., student burned a bridge).
- **Successful, but *task-irrelevant* action.** An action that does not limit future success may reflect a misconception in the student model if it does not advance the student down a path of success. We assume the student is performing tasks with an intention of reaching the goal. If an action does not result in progress toward that goal, perhaps the student holds mistaken beliefs.
- **Successful, plan-relevant, but *suboptimal* action.** Even though an action succeeds and is on a path toward the goal, it may introduce problems that result in a less-optimal plan (e.g., one that requires more actions than would otherwise be required).
- **Successful, plan-relevant, optimal, but *threat-ignorant* action.** An action that will be required as part of any successful plan, but does not resolve an obvious, immediate threat may indicate a gap or error in the student model. Perhaps the player is unaware of the sniper, because he is reloading his gun instead of scanning the roof tops for muzzle flashes.
- **Successful plan-relevant, optimal, threat-aware action.** In this situation, the student has chosen an optimal action.

- **Inaction.** This is the case where the student has not performed any action.

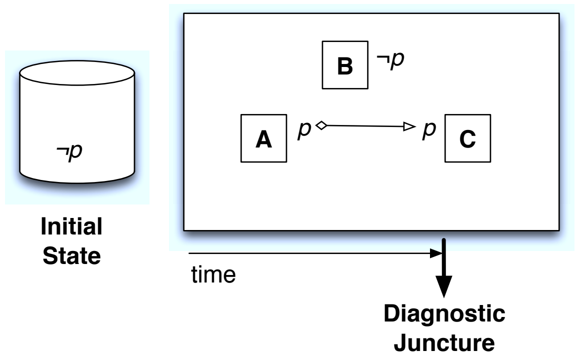


Fig. 7. Unresolved Threat

Bug Source	ID	Nickname	Propositional Description
Operator Schema	2A	Unknown Effect	$\neg$**Bel(has-effect(B, $\neg$*p*))**
Plan History	2B	Sequencing Error	**Bel (time(A) $\succ$ time(B)** $\wedge$ $\neg$ (time(A) $\succ$ time(B))
Plan History	2C	Unobserved Threat	**Bel (time(B) $\succ$ time(C)** $\wedge$ $\neg$ (time(B) $\succ$ time(C))
Plan History	2D	Phantom Intervention (action X has not happened)	**Bel (has-effect(X, *p*))** $\wedge$ **(has-effect(X, *p*))** $\wedge$ Bel (time(X) $\prec$ time(C)) $\wedge$ Bel (time(B) $\prec$ time(X)) $\wedge$ $\neg$ (time(X) $\prec$ time(C))
Operator Schema	1A	Unknown Precondition	$\neg$**Bel(has-precondition(C, *p*))**

TABLE 1
Action Failure - Unresolved Threat

6.9.2 *Diagnostic Templates: Sources of Belief*

Each student belief in a diagnostic template is classified according to which of the five components of the student model it affects: the **Operator Schema**, the **Initial State**, the **Goal State**, the **Plan History**, and the student's understanding of future possibilities described by the **Plan Future**. For the diagnostic template depicted in Table 1, the first misconception concerns the **Operator Schema**, and the remaining four misconceptions correspond to errors in the student's understanding of the possible **Plan Future**.

6.9.3 *Student Model: Updating Beliefs*

Each time a student-initiated action succeeds or fails, Annie considers all of these scenarios in determining which beliefs should be updated in the student model. Where the scenarios describe positive beliefs, Annie will strengthen the associated belief in the student model. Where the scenarios describe negative beliefs, Annie will weaken the associated belief strength in the student model.

6.10 Execution Cycle: Example Walkthrough

To help summarize the execution cycle, we return to the FixIt evaluation environment. By the time the third mission in this game begins, the student will have killed a malicious process, and killed a process that is being spawned by another process. Annie will likely reflect in its student model that the student has a higher than neutral likelihood of knowing many of the preconditions and effects of the kill-process operator. The third mission adds a new wrinkle, in that the malicious child process is spawned after each restart by a hidden startup file that infects a trusted system program. A successful path through this mission requires the student to complete the following tasks in order:

1) Kill the harmful child process.
2) Delete startup file that infects the parent process.
3) Kill parent process (child process' hidden re-spawner)

With Annie's help, this example might play out as follows. As the mission begins, we would expect Annie to choose not to intervene, given that the student model already shows a high likelihood that the student knows what is required for killing processes and the first job of the student is to kill a process. The student would probably kill the child process, and Annie would still not intervene, since the student model is initialized with a neutral likelihood that the student knows about deleting hidden startup files. In light of Annie's inaction, the student's next action would be to kill the parent process. As noted, this triggers a restart, since the parent is a trusted system program, and after the restart, the child is re-spawned. However, as Annie processes the result of this action, it will update the student model to reflect that the student probably does not know about the hidden file that must be deleted so that the parent will not be re-infected.

Following the restart, Annie may choose to fix this flaw in the student model with a remediation. Depending upon how the student has fared with other remediations, Annie may choose a subtle hint, like a diversion that directs attention toward the location that contains the hidden file, or it may choose something more direct, like a system-controlled character who approaches the player character and mentions that he has heard that someone spotted a trojan startup file on the system. If the student attempts to delete the file without first killing the parent process, then Annie may introduce another remediation to advise the student of the FILE-IN-USE flag as discussed earlier.

7 Experimental Evaluation

This section describes an initial experimental evaluation to assess the accuracy with which Annie is able to estimate human subjects' understanding within an exploratory learning environment.

7.1 Experimental Design

Sixteen adult college graduates from a variety of disciplines volunteered to participate in the study. 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. The experimenter provided no additional information to the subject. The subject was asked to read this set of instructions with no explicit time limit given. 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.

7.2 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 was trained for several hours in the use of the game, including all the potential actions and learning objectives 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, we had 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. We 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 2. 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 ($p < 0.0001$). This means that we found 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.

Subject Code	**Diagnostic Accuracy**	
	Annie's Student Model	**Expert Human Observer**
Subject 1	91.67%	91.67%
Subject 2	75.00%	70.83%
Subject 3	87.50%	83.33%
Subject 4	87.50%	91.67%
Subject 5	79.17%	91.67%
Subject 6	95.83%	91.67%
Subject 7	70.83%	75.00%
Subject 8	75.00%	79.17%
Subject 9	66.67%	54.17%
Subject 10	66.67%	62.50%
Subject 11	58.33%	58.33%
Subject 12	70.83%	66.67%
Subject 13	70.83%	70.83%
Subject 14	95.83%	91.67%
Subject 15	41.67%	50.00%
Subject 16	83.33%	70.83%

TABLE 2
Per-Student Comparative Diagnostic Accuracy

Several factors contributed to Annie's strong performance. First, we were careful 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 to assess the student's knowledge as completely accurate in these cases, but such assessments would be useless, as there would be nothing left to teach. Rather, our aim was to allow only enough time for our students to learn an average 25-50% of the game content. Our rationale is that this is the level of understanding most germane to Annie's prospective deliberations. Secondly, we want the game to have a rather short duration, 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 is a higher probability in correctly predicting that the student is unknowledgeable about actions that they never performed. We will soon conduct an expanded evaluation of Annie in a much more complex game expected to engage each student for at least thirty minutes. An important question we hope to answer in that expanded study is whether Annie's student modeling can retain its accuracy over a longer duration in a task-rich environment.

8 Discussion and Conclusion

This paper describes a novel approach for building intelligent educational games by integrating pedagogy with the core mechanics of games. Our system, called Annie, uses plan-based representations to describe game actions, game states, and student knowledge. Annie leverages these representations to provide intelligent tutoring within games.

Annie is not sufficient by itself to ensure that an educational game is fun. Rather, it separates the tasks of designing for fun and designing for teaching in such a way that the results of these efforts can be automatically integrated. Although this relieves the burden on the tutorial designer of balancing fun with teaching, it demands that both learning goals and pedagogical content are expressed in plan-based terms, which will be easier for some learning domains than it is for others.

Our preliminary evaluation has found a strong and significant correlation between Annie and a human observer in diagnosing students' knowledge acquisition in an exploratory game environment. We will soon conduct a more extensive evaluation of Annie's teaching effectiveness within a more complex and lengthy game, where Annie's student model will be used at run-time to generate learner-specific guidance.

REFERENCES

[1] K. VanLehn. The Behavior of Tutoring Systems. *International Journal of Artificial Intelligence in Education*, 16(3):227–265, 2006.

[2] J.P. Gee. What video games have to teach us about learning and literacy. *Computers in Entertainment (CIE)*, 1(1):20–20, 2003.

[3] M. Prensky. Digital game-based learning. *Computers in Entertainment (CIE)*, 1(1):21–21, 2003.

[4] Marc Prensky and Jan-Cannon Bowers. Serious games debate. Serious Games Summit, October 2005.

[5] Chris Bateman. *Game Writing: Narrative Skills for Videogames (Charles River Media Game Development (Paperback))*. Charles River Media, Inc., Rockland, MA, USA, 2006.

[6] Maic Masuch. Is this the real thing? realism and fun in computer games. NSCU 'Future of Games' Lecture Series, August 2007.

[7] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper & Row USA;, 1990.

[8] Hal Barwood. The language of games. Austin Game Writers Conference, October 2005.

[9] L.S. Vygotsky. *Thought and Language*. MIT Press Cambridge, Mass, 1986.

[10] T. Murray and I. Arroyo. Towards Measuring and Maintaining the Zone of Proximal Development in Adaptive Instructional Systems. *ITS*, 2002.

[11] V.J. Shute and R. Glaser. A Large-Scale Evaluation of an Intelligent Discovery World: Smithtown. *Interactive Learning Environments*, 1(1):51–77, 1990.

[12] A. Bunt and C. Conati. Probabilistic Student Modelling to Improve Exploratory Behaviour. *User Modeling and User-Adapted Interaction*, 13(3):269–309, 2003.

[13] B.W. Mott and J.C. Lester. U-director: a decision-theoretic narrative planning architecture for storytelling environments. *AAMAS*, pages 977–984, 2006.

[14] Ruth Aylett, Sandy Louchart, Joao Dias, Ana Paiva, Marco Vala, Sarah Woods, and Lynne Hall. Unscripted narrative for affectively driven characters. *IEEE Comput. Graph. Appl.*, 26(3):42–52, 2006.

[15] C. Quintana, B.J. Reiser, E.A. Davis, J. Krajcik, E. Fretz, R.G. Duncan, E. Kyza, D. Edelson, and E. Soloway. A Scaffolding Design Framework for Software to Support Science Inquiry. *The Journal of the Learning Sciences*, 13(3):337–386, 2004.

[16] T. de Jong. Technological advances in inquiry learning. *Science(Washington, D. C.)*, 312(5773):532–533, 2006.

[17] Bernard H. Suits. *The grasshopper: Games, life and utopia*. Broadview Press, 2005.

[18] Katie Salen and Eric Zimmerman. *Rules of Play : Game Design Fundamentals*. The MIT Press, October 2003.

[19] R. Koster and W. Wright. *A theory of fun for game design*. Paraglyph press, 2004.

[20] A. Rollings and E. Adams. *Andrew Rollings and Ernest Adams on game design*. New Riders Games, 2003.

[21] R. Fikes and N. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intellligence*, 2(3/4), 1971.

[22] M. Mateas and A. Stern. Structuring Content in the Façade Interactive Drama Architecture. *AIIDE*, pages 93–98, 2005.

[23] M. Cavazza, F. Charles, and S.J. Mead. Interacting with virtual characters in interactive storytelling. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 318–325, 2002.

[24] Mark Riedl, C.J. Saretto, and R. Michael Young. Managing interaction between users and agents in a multi-agent storytelling environment. In *AAMAS*, 2003.

[25] Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167 – 238, 1995. Planning and Scheduling.

[26] C. Thompson. Halo 3: How microsoft labs invented a new science of play. *Wired. Retrieved*, 31(2008):15–09, 2007.

[27] C. Bateman. *Game writing: Narrative skills for videogames*. Charles River Media, 2007.

[28] C. Crawford. *The art of computer game design*. Osborne/McGraw-Hill, 1984.

[29] T.M. Vernieri. A Web Services Approach to Generating and Using Plans in Configurable Execution Environments. Master's thesis, North Carolina State University, Raleigh, North Carolina, January 2006.

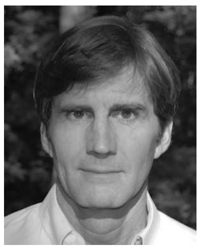
[30] J.M. Thomas and R.M. Young. Author in the loop: Using mixed-initiative planning to improve interactive narrative. *Workshop on AI Planning for Computer Games and Synthetic Characters*, 2006.

[31] J. Skorupski, L. Jayapalan, S. Marquez, and M. Mateas. Wide ruled: A friendly interface to author-goal based story generation. *LECTURE NOTES IN COMPUTER SCIENCE*, 4871:26, 2007.

[32] J.M. Thomas and R.M. Young. Using Task-Based Modeling to Generate Scaffolding in Narrative-Guided Exploratory Learning Environments. *Proceedings of the 14th International Conference on Artificial Intelligence in Education*, 2009.

[33] R.C. Murray and K. VanLehn. A Comparison of Decision-Theoretic, Fixed-Policy and Random Tutorial Action Selection. *LECTURE NOTES IN COMPUTER SCIENCE*, 4053:114, 2006.

[34] R.M. Young, M.E. Pollack, and J.D. Moore. Decomposition and causality in partial-order planning. *Proceedings of the Second International Conference on AI and Planning Systems*, 48, 1994.



James M. (Jim) Thomas James M. Thomas (M'10) received the B.A. degree in mathematics in 1983, and the M.Eng. degree in computer science in 1986, both from the University of Virginia in Charlottesville, VA. He is currently working toward the Ph.D. degree in computer science at North Carolina State University, in Raleigh, NC as an NSF Graduate Research Fellow.

He has worked at IBM, BNR, Nortel Networks, and 3-C ISD. His research interests include intelligent tutoring systems, plan-based knowledge representation, social skills training, and games-based learning.

R. Michael Young R. Michael Young (M'98–SM'07) received the B.S. degree in computer science from the California State University at Sacramento, in 1984, the M.S. degree in computer science from Stanford University, in 1988, and the Ph.D. degree in intelligent systems from the University of Pittsburgh, in 1998.

He has worked at Hewlett-Packard, FMC Corporation, Rockwell, and as a Postdoctoral Fellow at Carnegie Mellon University. He is currently an Associate Professor of Computer Science and Co-Director of the Digital Games Research Center, North Carolina State University, Raleigh, NC. His research interests include the computational modeling of narrative, planning, games-based learning, and automated cinematography.