EVOLVING EXPERT KNOWLEDGE BASES:

APPLICATIONS OF CROWDSOURCING AND SERIOUS GAMING TO ADVANCE KNOWLEDGE DEVELOPMENT FOR INTELLIGENT TUTORING SYSTEMS

A Dissertation Presented

By

MARK FLORYAN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2013

School of Computer Science

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

EVOLVING EXPERT KNOWLEDGE BASES:
APPLICATIONS OF CROWDSOURCING AND SERIOUS GAMING TO ADVANCE KNOWLEDGE DEVELOPMENT FOR INTELLIGENT TUTORING SYSTEMS

A Dissertation Presented

By

MARK FLORYAN

Approved as to style and content by:

_____
Beverly Park Woolf, Chair

_____
Robert Moll, Member

_____
W. Richards Adrion, Member

_____
Florence Sullivan, Member

_____
Merle Bruno, Member

_____
Lori Clarke, Chair
School of Computer Science

DEDICATION

To my Mom and Dad

ACKNOWLEDGMENTS

I would like to first thank my advisor Beverly Woolf for her unwavering and positive support over the years. I will always appreciate how well Beverly balances positive support with stark criticism in such a way that kept me improving my craft while also remaining optimistic, excited, and focused. I also want to thank my committee members, Richard Adrion, Robert Moll, and Florence Sullivan, for their input and criticism on this work. This work could not have been done without Merle Bruno, who painstakingly constructed the human expert knowledge base over the years for Rashi. Additionally, this work would not have been possible without the support and flexibility of the UMass Biology Department. I would like to thank Robert Cairl, his students, and his teaching assistants for providing us with a plethora of data over the years.

I would also like to thank Toby Dragon and Tom Murray; the other members of the Rashi team from whose wisdom I have greatly benefitted. Lastly, I would like to thank all of the individuals in the Center for Knowledge Communication with whom I have had many conversations regarding this work and intelligent tutoring systems in general. Every one of these conversations helped me grow intellectually and helped make this work possible.

ABSTRACT

EVOLVING EXPERT KNOWLEDGE BASES

APPLICATIONS OF CROWDSOURCING AND SERIOUS GAMING TO ADVANCE KNOWLEDGE DEVELOPMENT FOR INTELLIGENT TUTORING SYSTEMS

MAY 2013

MARK FLORYAN, B.S., UNIVERSITY OF VIRGINIA

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Beverly Park Woolf

The state of modern computing technology has presented powerful opportunities to enhance education through the applications of artificial intelligence. These opportunities have led to the emergence of research in Intelligent Tutoring Systems (ITS) [103] in which Artificial Intelligence (AI) is applied within computer programs that guide, scaffold, and instruct students at a competency equal to that of a human teacher [55]. These ITS technologies have been employed in many domains, including those that are well defined like mathematics [6] and those that are ill-defined such as law [2].

This dissertation presents a novel effort to develop ITS technologies that adapt by observing student behavior. In particular, we define an evolving expert knowledge base (EEKB) that structures a domain's information as a set of nodes and the relationships that

exist between those nodes. The structure of this model is not the particularly novel aspect of this work, but rather the model's evolving behavior. Past efforts have shown that this model, once created, is useful for providing students with expert feedback as they work within our ITS called Rashi [34, 35, 109]. We present an algorithm that observes groups of students as they work within Rashi, and collects student contributions to form an accurate domain level EEKB. We then present experimentation that simulates more than 15,000 data points of real student interaction and analyzes the quality of the EEKB models that are produced. We discover that EEKB models can be constructed accurately, and with significant efficiency compared to human constructed models of the same form. We are able to make this judgement by comparing our automatically constructed models with similar models that were hand crafted by a small team of domain experts.

We also explore several tertiary effects. We focus on the impact that gaming and game mechanics have on various aspects of this model acquisition process. We discuss explicit game mechanics that were implemented in the source ITS from which our data was collected. Students who are given our system with game mechanics contribute higher amounts of data, while also performing higher quality work. Additionally, we define a novel type of game called a knowledge-refinement game (KRG), which motivates players to contribute to an already constructed EEKB, but for the purpose of refining the model in areas in which confidence is low. Experimental work with the KRG provides strong evidence that: 1) the quality of the original EEKB was indeed strong, as validated by KRG players, and 2) both the quality and breadth of knowledge within the EEKB are increased when players use the KRG.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

OVERVIEW OF KNOWLEDGE ACQUISITION WITHIN TUTORS

Intelligent tutoring systems have proven themselves, on numerous occasions, as effective learning tools [6, 111, 112]. However, the development of such systems is often cumbersome, requiring the tedious effort of a few experts. In particular, intelligent tutoring systems require copious amounts of content development [67], a problem that is heightened when a system must provide intelligent feedback or support. In this dissertation, we discuss techniques for providing an intelligent agent that observes student behavior and infers an accurate domain level knowledge base, while also evaluating methods for optimizing the build time efficiency, quality, and breadth of this knowledge base.

This project aims to articulate and provide evidence for three broad claims. Firstly, we aim to show that an amalgamation of student work within a tutor, given enough students, is sufficient for constructing quality domain level knowledge bases and that this process can be done algorithmically. Secondly, we argue that our automatic knowledge acquisition process outperforms competing methods with regard to efficiency. Lastly, we posit that incorporating game mechanics into various aspects of this process increases its efficacy by increasing student engagement within a tutor, encouraging focused student input, and by providing mechanisms for experts to quickly refine created models.

Our approach is to invite students to contribute knowledge by utilizing an intelligent tutor with game-like mechanics. The tutor in question invites students to accept the role of

doctor and diagnose a patient who has become ill. To this end, we utilize the Rashi intelligent tutoring system [33]. We built an algorithm in the Rashi system that automatically estimates expert domain knowledge by analyzing student input to the tutor. In addition, our human domain expert has hand crafted vast knowledge bases for Rashi cases manually. Thus we can provide evidence, from related work [34, 35], that obtaining these domain models is useful. In addition, these human generated models are useful for comparison with our automated ones to judge quality.

Our results show that algorithms for knowledge base generation are effective, but still erroneous in places. We present a new type of game called 'knowledge-refinement games' (KRGs), the intention of which is to support experts to refine the knowledge base in places that our algorithm was not confident. The game presents human players with short propositions that have been inferred from student data. The player is invited to view these and simply enter true or false, after which the system automatically adjusts and refines the knowledge base. Game mechanics are incorporated to promote necessary incentives. For example, the game keeps high score totals to motivate players to succeed and to continue playing. In addition, game mechanics were used to incentivize asynchronized players to try to produce similar answers as a way to prevent players from 'gaming' the system.

Thus, we present a process for increasing the overall automation, adaptability, and independence of intelligent tutoring systems in ill-defined domains by comparing several disparate but related experiments. We posit that designers of tutoring systems can focus

on producing systems with a minimum set of intelligent features. We then argue that the incorporation of simple game mechanics, as well as our presented (or similar) algorithm can effectively provide the means for a system to learn from contributors (students) and utilize this learned material to provide intelligent feedback and dynamic support to future students.

1.1 An Overview of our Algorithm

Our algorithm automatically constructs an evolving expert knowledge base (EEKB)[1]. Our EEKB is represented by a set of nodes N and a set of relationships R = {r= (n1, n2)}, where n1 and n2 are members of the set of nodes N. Additionally, each node is annotated with domain information. In the medical diagnosis domain, nodes can be annotated as hypothesis/diagnosis conditions (hyperthyroidism, allergy, etc…), or as data/evidence (blood iron level is high). Likewise, relationships between nodes are annotated with a semantic description of the relationship (for example, blood iron level is high SUPPORTS hyperthyroidism).

Within our tutoring system Rashi, students are required to diagnose ill patients by demonstrating that they understand hypotheses and evidence within the medical diagnosis field, as well as the relationships between hypotheses and evidence. Specifically, students use the system to create an argument that, if done well, should reflect a subset of our human crafted expert knowledge base (HEKB).

[1] *For the rest of this document, we use expert knowledge base (EKB) to refer to any generic knowledge model constructed in any way, and evolving expert knowledge base (EEKB) to refer specifically to an EKB created by our automatic algorithm.*

Thus, our algorithm analyzes student created nodes, and relationships, and accepts this information for consideration as "true" expert knowledge with a default confidence. As we observe students considering information in multiple instances, the algorithm increases its confidence in particular EEKB entries. Once past a confidence threshold, the data is included as true information within the EEKB.

Of course, students are not experts, and thus our algorithm takes several additional factors into account. These issues include inaccuracy, misconceptions, repetitive information, incomplete information, and contradictory information.

1.2 Judging our Algorithm's Efficacy?

To judge the quality of our evolving expert knowledge base, we compare it directly to our human created knowledge base (which is preserved for the purpose of performing this comparison). We utilize two distinct but related metrics. The first, called 'precision' judges the truth of the generated knowledge with little regard to its breadth. The second, called 'recall, is the same as precision but takes into account the breadth of knowledge that is created. This terminology is borrowed from the research area of information retrieval[2] because of the similarities in definition and application.

We define precision as the percentage of entries in the automatically generated graph that also occur in the human graph. Thus, the precision of a generated graph will be 1.0 (100 percent) if the generated graph is any subset of the human generated graph. The formula

[2] http://en.wikipedia.org/wiki/Precision_and_recall
http://en.wikipedia.org/wiki/Information_retrieval

for assessing precision of a generated graph (EEKB) compared to a human created graph (HEKB) is:

$$Precision\ (EEKB, HEKB) = |\ EEKB \cap HEKB\ |\ /\ |\ EEKB\ |$$

Where EEKB is the automatically generated graph and HEKB is the human generated graph

The cardinality, expressed in the formula above, of any given graph includes both the set of nodes and relationships of the graph in question. This calculation is, of course, predicated on the assumption that an expert created graph is always filled with accurate information.

Recall is the percentage of the human expert's knowledge base that we have successfully generated. Thus, this is a measurement of how well we have included the breadth of knowledge created by a human expert.

$$Recall\ (EEKB, HEKB) = |\ EEKB \cap HEKB\ |\ /\ |\ HEKB\ |$$

Where EEKB is the automatically generated graph and HEKB is the human generated graph

1.3 An Overview of our Knowledge-Refinement Game

Because we are analyzing the responses of students (who are not domain experts), our automated knowledge base is somewhat inaccurate. To help refine the knowledge, we created a game to be played by human experts or 'pseudo-experts'. The game pulls small pieces of knowledge from our generated EEKB, and presents them to the expert as a question with two answer choices. The expert responds, and the EEKB is automatically refined with increased confidence. Some example questions within this game include.

1. Do these two nodes refer to the same piece of knowledge? Expert answers yes or no. If so, combine the nodes.

2. Examine these two nodes and their relationship. Is this relationship correct? Expert answers yes or no.

3. These statements contradict one another, which of them is correct? The expert chooses one of the statements.

More question types are included. Expert players may answer as few or as many questions as they like. In addition, we track the changes in precision and recall for the generated EEKB as this refinement game is being played to analyze its efficacy.

1.4 Proving The Efficacy of our Approach

Lastly, we aim to compare the efficiency of our approach against that of a human expert creating knowledge by hand. The human expert estimated the time spent on our current human generated knowledge base. We compare this to an adjusted measure of time for constructing the automated knowledge. This measure will incorporate the accuracy and correctness of knowledge bases that are created by each method. We then make quantifiable claims about the efficiency of our approach in terms of the breadth and accuracy of knowledge that can be obtained per hour. Our measurement is calculated in the following way.

$$KnowledgeEfficiency\ (EEKB) = |\ trueNodesAndRelations\ (EEKB)\ |\ /\ numHours(EEKB)$$

Where trueNodesAndRelations(EEKB) is a function returning the number of elements in the EEKB that are considered to be accurate domain information and numHours(EEKB)

is the number of hours necessary to create the EEKB. Thus, the efficiency of building a knowledge base is simply the size of the true propositions within that knowledge base divided by the time it takes to build it.

1.5 Summary of this Document

The remainder of this document will describe, in detail, this dissertation with two overall goals in mind. The first goal is to verify the feasibility of automatically constructing knowledge bases by observing student interactions, and by analyzing usage of a novel knowledge refinement game. The second goal` is to provide evidence that such approaches are not only effective, but are bolstered by the application of game design mechanics.

Chapter two describes the related literature in detail. Chapter three describes the design of the relevant systems and algorithms utilized for our studies. We then describe our research design in Chapter 4, along with our results in Chapter 5. Finally, we conclude by discussing the results and future work in Chapter 6.

CHAPTER 2

INTELLIGENT TUTORS, SERIOUS GAMES, AND AUTHORING TOOLS

In this chapter, we review relevant literature from a variety of fields to help inform our research design. We garner and coalesce this information from a variety of independent fields in an attempt to unify some of the disparate philosophies and principles. We begin by discussing pedagogies for ill-defined domains, particularly because we intend to prove our approach within the construct of medical diagnosis, an example ill-defined domain. We then discuss the broader field of intelligent tutoring systems in which software applications and artificial intelligence are applied to instruct students effectively in a range of domains. This helps us calibrate the efficacy with which artificial intelligence is useful within computerized tutors, and improves our understanding of the differences between designing tutors for different domain. We give particular focus to the subfield of serious games, in which ITS technologies are outfitted with game mechanics in an attempt to improve their efficacy. Most of the research in this area focuses on the improvements to student cognition and learning when game mechanics are implemented. We argue that these studied effects can be incorporated to improve the quantity and quality of student contributions to a knowledge acquisition algorithm.

We then acknowledge the active field of authoring tools as relevant to this work. Authoring tools are programs intended to make the construction of intelligent tutoring systems efficient and viable by non-programmers. Advances in this field are relevant in that our approach, although using automatic means, is intended to improve methods for automatically constructing aspects of tutoring systems. Lastly, we discuss the emerging

field of crowdsourcing, in which multiple anonymous contributors provide collected work that is requested. We specifically focus on 'games with a purpose' (GWAPS) in which games are developed for such purposes. We observe that crowdsourcing is effective in many applications, but has yet to leverage the work of school students to develop artificial intelligence models.

2.1 Overview: What fields are we talking about here?

The theory informing the work for this dissertation spans four distinct fields, focusing on specific sub-fields in some cases. The relevant literature belongs in the following categories:

1) *Pedagogy for Ill-Defined Domains:* Ill-defined domains are those in which relevant problems do not contain one set solution or solution path [4]. Because knowledge in these domains is often best modeled as semantic graphs, they are well suited for testing our approach.

2) *Intelligent Tutoring Systems / Serious Games:* Intelligent Tutoring Systems are computer applications that attempt to teach a subject at or above the competency of a one-on-one human tutor. These systems often involve the application of artificial intelligence algorithms to simulate intelligent reactive behavior [6]. Our review of this field will focus mostly on systems that employ Expert Knowledge Bases [33], and systems that incorporate gaming mechanics [77].

3) *Authoring Tools for Intelligent Tutors:* This sub-field of intelligent tutoring systems involves the development and use of tools that aid in the

efficient and timely production of computerized tutors. These tools often allow non-programmers to create tutoring systems with relative ease [65].

4) *Crowdsourcing (specifically GWAPS):* This nascent field involves the accomplishment of any of a variety of tasks via the gathering of data from a large host of users. Many crowdsourcing applications involve the division of labor amongst a plethora of volunteers [5], but other applications involve the gathering of useful information via the actions of game players [39].

We begin by discussing the most relevant literature in the field of ill-defined domains.

2.2 Pedagogy for Ill-defined Spaces

Because this work focuses on knowledge base development in medical diagnosis, it is important to define the characteristics of such ill-defined domains. Ill-defined domains are those in which a single solution or solution path does not exist [3]. For example, the application of the law is ill-defined, because the law can often not be interpreted in a strictly logical manner, but rather requires the subtle interpretation of the intent of written law [2]. Other ill-defined domains include teaching literature, art history, human biology, and forestry. Several researchers [117][3] have identified various attributes of ill-defined domains including but not limited to:

- A lack of widely accepted domain theories.
- The need to reason analogically with cases or examples.
- Large solution spaces that prohibit the enumeration of possible solutions
- A lack of clear criteria for judging solutions.
- Are not considered solved when a single solution is proposed.

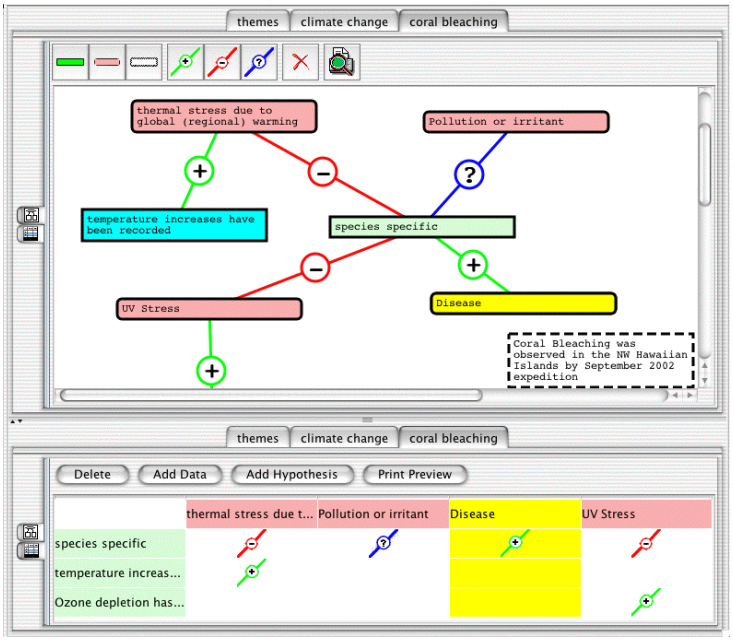


Figure 2.1: An image of the Belvedere software, in which students map out concepts and ideas on a canvas with nodes connected with relationships.

Tutors have been created that are intended to instruct students on topics in ill-defined domains. However, the student tutor interaction is often more open-ended and requires a more probabilistic model of the student's work. For example, Belvedere is a system that allows students to work within generic problem based environments [92] and provides a vast array of knowledge visualizations to assist students in formulating their ideas.

As mentioned earlier, another ill-defined domain of interest is law. Vincent Aleven et al. have created the 'Hypothesis Formation' project, to show how visualization of argument graphs helps law students improve their argumentation skills [1]. This is done by providing students a forum within which students can organize their arguments and thus visualize areas of their argument that require further support along with other necessary

additions. The system also includes on-demand feedback and prompting to intelligently support student's efforts.

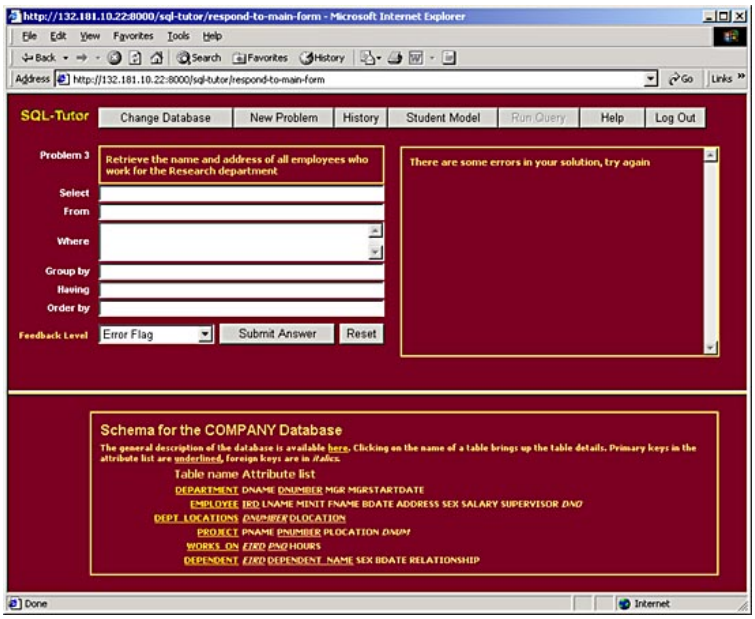


Figure 2.2: A screenshot of the SQL-Tutor, which uses a constraint-based model to teach students to use an SQL database.

There are several ways to handle modeling of the knowledge in ill-defined domains. One such approach, coined by Mitrovic et al., is using constraint-based tutors [64] that define a domain via constraints in the domain that must not be violated while solving problems in that domain. Because of this, a tutor will not be able to solve a problem internally, but is able to describe the violated constraints of a faulty solution. Constraint-based tutors do not require executable expert modules. Examples of constraint-based tutors include the SQL-Tutor, Capit (punctuation tutor) [64], and Kermit (Database design tutor) [65].

The research efforts described in this dissertation involve knowledge development in the field of medical diagnosis. This field is considered ill-defined because of uncertain

nature of evidence, and the results that evidence implies. Although medical diagnosis is a scientific domain, the findings that are applied are often updated, and even experts are likely to disagree on complex cases. Thus, we attempt show in this work that the complexity of medical diagnosis (and thus ill-defined domains) can be constructed automatically through a mass quantity of student analyses in a computationally readable format. We, however, do not wish to imply that our approach is limited to ill-defined domains. In fact, we find it likely that our approach is reasonable in many other domains for which intelligent tutors are implemented. We next explore the field of intelligent tutoring systems, with emphasis on how our approach may be applied to or bolstered by the advances in ITS.

2.3 Intelligent Tutoring Systems

The second field we describe is intelligent tutoring systems, or computer programs that attempt to simulate the experience and (more importantly) the efficacy of learning from a human tutor [103]. It is well known that the typical fifteen to one student to teacher ratio is an ineffective strategy for learning [107]. In addition, it is also well documented that a one-to-one teacher student interaction is enormously more effective, leading to statistically significant learning gains [108]. The main obstacle then, is that there exist many more students in need of individual tutoring than human tutors capable of spending extended time with them.

This is where intelligent tutors aim to provide powerful advantages. With a well-designed intelligent tutor, students can work within virtual environments that instruct in many

domains, including Mathematics, Physics, Law, and Human Biology. Many of these tutors employ advanced artificial intelligence techniques to foster realistic interactions [7].

For example, two competitive mathematics tutors include the Carnegie Learning Online Math Tutor, founded by scientists at Carnegie Mellon University [55] and Wayang Outpost, a math tutor developed at the University of Massachusetts, Amherst [6]. These tutors invite students to practice a plethora of math problems. While working within these systems students receive dynamic hints and individualized feedback, and interact with affective pedagogical agents.

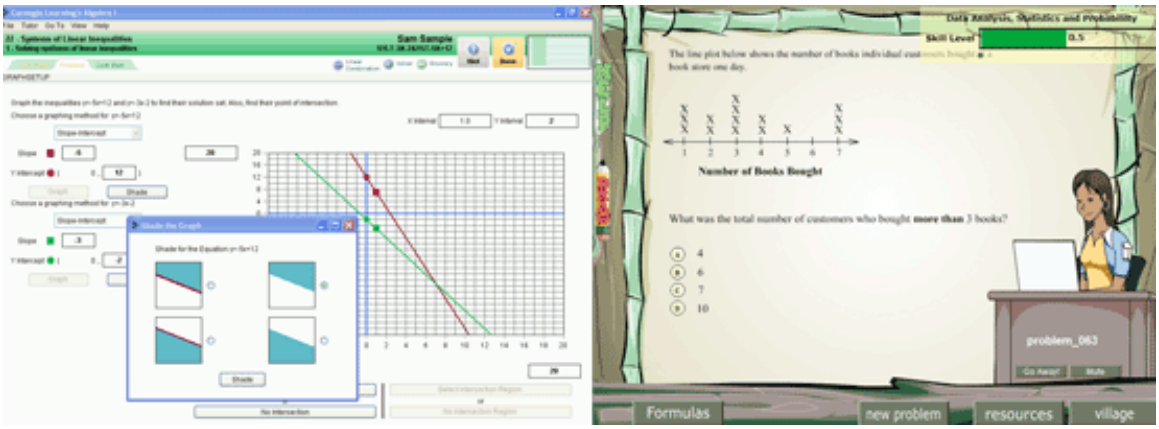


Figure 2.3: The Carnegie Learning Cognitive Tutor (left) and the Wayang Outpost Mathematics Tutor (right). Both of these tutors teach Mathematics skills.

2.2.1 Expert Systems

An expert system is a computer program that reasons about a problem in much the same way, and with about the same performance, as do specialists. Knowledge-based expert systems, or simply expert systems, use human knowledge to solve problems that

normally would require human intelligence. These expert systems represent the expertise knowledge as data or rules within the computer. Knowledge-based systems collect the small fragments of human know-how into a knowledge-base which is used to reason through a problem, using the knowledge that is appropriate. Constraint-based models (CBM), described in the previous section, are an example of expert systems. However, constraint-based tutors model the constraints of knowledge instead of the knowledge itself. CBM's are useful, but cannot solve problems themselves. In this section, we focus on systems that use a different form of expert system and model knowledge directly, so that intelligent reasoning can be computed algorithmically.

For example, GUIDON [19] is an ITS for learning diagnosis of infectious diseases. The diagnosis is determined by a patient's prior medical history, and utilizes a rule-based expert system called MYCIN. MYCIN contains several hundred rule-based, domain level entries that help support student efforts or produce expert solutions. A rule-based expert knowledge base contains entries of a specific form, notably condition action pairs that lead to a consequence. For example, one rule might be that if a patient is losing weight and a blood test shows a low iron count, then the patient has a thyroid problem.

Another popular form of expert model, a model based on the fundamental knowledge of a domain, is one built with probabilistic approaches. These models are generally computational graphs whose nodes and edges are accepted as true with an annotated probability. In intelligent tutoring, these models are often used to predict the level of knowledge that the student has learned. For example, 'Andes' is a tutoring system that

teaches physics through sets of practice problems [113]. The system contains powerful visualizations, a workspace that is highly interactive, and a feedback system.

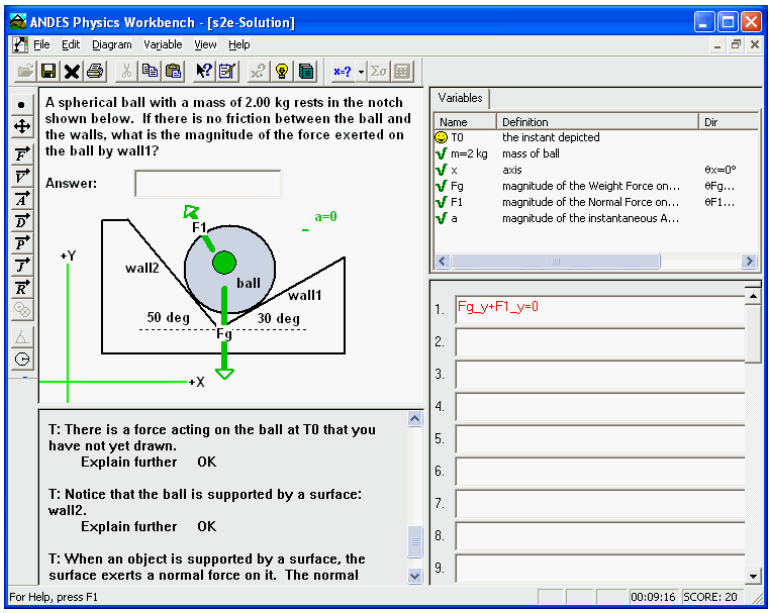


Figure 2.4: A screen shot from the Andes Physics Tutor.

Andes contains a type of probabilistic model called a Bayesian Net. A student's potential action paths for a given topic are modeled out ahead of time. As a student works within the system, the probabilities that a student knows a topic are dynamically updated from evidence provided. Additionally, probabilities are propagated through edges because having knowledge of a particular node is evidence that nearby nodes have been learned as well. Probabilistic models are a strong way to dynamically alter a model of a student's understanding. However, it is important that the model be complete and accurate, which involves measuring the accuracy of the probabilities model in predicting human learning. Models such as those found in Andes are of the form that we wish to construct automatically, to alleviate the need for domain experts to construct them tediously.

However, in order to construct an expert knowledge base automatically, a system needs a mechanism for observing and evaluating student work within a tutor. Useful research for application in this regard is coined by Valerie Shute, and is called stealth assessment [81]. For example, games that induce 'flow' [26] are important for learning, but active assessments within those games WILL cause a disruption in this flow. Thus, stealth assessments are needed that do not disrupt the user from the learning activity. Stealth assessment is about embedding student skills as variables (like health or points) and allowing students to know that by engaging in proper activities, these scores can be raised. In essence, the strategy is to indirectly impact the behavior of a student in a positive way.

Evidence Centered Design (ECD), a model in which student actions inform various pieces of "evidence" regarding their state, [80] and Bayes Nets can be used for stealth assessment. However, the system must elicit behavior that informs assessment. The following models are thus used to create a system for use with ECD:

> ***Evidence Model:*** *a model of how learner actions provide evidence of their knowledge (a student model is one example).*
>
> ***Competency Model:*** *a model of the knowledge (or competencies) that students are learning.*
>
> ***Task Model:*** *a model of tasks that relate to the skills that students are learning.*

Evidence Centered Design has been used to assess the value of playing certain games. For example, Shute et al., used ECD to assess the impact on learning the physics puzzle game World of Goo [82]. The researchers observed players attempting to solve puzzles within the game.

Figure 2.5: An example level in World of Goo. Players must build a bridge, held up by balloons, that neither sinks too low (bridge touches spikes) nor is raised too high (balloons pop on upper spikes).

For example, one player scored better on the analyzed competencies than did another; but failed to solve the problems within the game itself. Conversely, a "weaker" player scored low on the analyzed competencies despite completing the level within the game. Further analysis showed that the stronger player was more exploratory and displayed more meta-cognitive knowledge, e.g. had knowledge of how to learn. Systems may need to have some kind of intelligent feedback model to support student's ability to use meta-cognitive knowledge. Not surprisingly, just winning the game is not the most important factor for learning in a video game. Although this research was focused on a game, the result can most likely be applied to any learning task, and thus reaching a 'goal' is not always sufficient for learning.

Another form of expert systems is recommender systems, or a computational system that predicts the 'rating' or 'preference' that a user would give to an item (such as music, books, or movies) or social element (e.g. people or groups) they had not yet considered. Such systems may provide suggestions for items to be of use to users using a model built from the characteristics of an item (content-based approaches) or the user's social environment (using collaborative filtering approaches)[3]. "Item" is the general term used to denote what the system recommends to users. These systems typically create user models based on long-term student action observation while creating expert systems by pooling together the knowledge of many individuals [59]. This is based on the idea that utilizing the knowledge of many individuals is useful, and may be grounds for fostering collaboration. Recommender systems have many commercial applications as well, being utilized by websites such as Netflix, Amazon, and YouTube.

Although recommender systems involve the creation of expert knowledge, it is unclear how much of this knowledge creation is simply the compilation of user opinions. These systems combine user data to gain a picture of what other group members are doing. Thus, recommender systems generally benefit from the fact that user input is always true (e.g. Netflix users will generally know whether they liked a movie or not) and that the knowledge that is being modeled is not truth (but rather reflects the beliefs of users). However, recommender systems are subject to inconsistencies, e.g., when more than one person within a single Netflix account selects movies and the model of the user becomes distorted. Recommender systems are distinctly different from the proposed dissertation

[3] *http://en.wikipedia.org/wiki/Recommender_system*

work because of the nature of the knowledge that is being modeled. Users of intelligent tutors do not always know whether their input is true (they are learning themselves), but the knowledge they are attempting to articulate is factual in nature. Thus, the model that is being learned must approach a level of truth as would be agreed upon by human experts.

Lastly, the Rashi system [33] is an example of a domain-independent tutor that teaches within ill-defined domains and utilizes an expert knowledge. Rashi is the platform for the experimentation conducted in this work. The system is described in more detail in Chapter 3. Rashi is also considered to be a serious game, as discussed in the next section. We believe that the benefit of serious games can increase the quality of student input, or at the very least keep students motivated and interested.

2.2.2 Serious Games

Many researchers have looked into the promise of using video games as educational tools. This new field has been coined "serious games". It is our goal to prove that serious game play can help provide the evidence necessary to automatically construct support knowledge bases. In this section, we discuss the benefits of serious games, and summarize several systems published recently.

Research shows that the most important categories for engagement in games are fantasy, challenge, and fun [52]. In general, game players do not wish to conquer the same problems in a game that they encounter in everyday life. Games like "World of

Warcraft" and "Second Life" have become popular in part because they allow players to immerse themselves in a new reality in which their existence is freely defined.

The challenge then for educational game designers is to define frameworks that help designers choose goals for players that optimize fantasy, challenge, and fun while not sacrificing academic focus. "Mathblaster" does this by creating a storyline (though a relatively vague one) [61]. One such storyline is that players traveling through space need to save a friend from an evil alien. "SimCity 2000" on the other hand, invites players to assume the role of a mayor of a futuristic city. Players make executive decisions in an attempt to lead their small city to prosperity [89]. Both of these games succeed in developing entertaining goals for students.

Various developers use varying motivations, theories, and principles when designing serious games, and each games has different advantages that can be analyzed to inform future serious games research. Two categories of serious games dominate the research, coined as exogenous and endogenous games [45]. Exogenous games are those in which game mechanics are disjointed from the game play. 'Math Blaster', a math game set in a futuristic space world, is an exogenous game because players are given story elements and visual effects that are interrupted for worksheet like math questions [61]. On the other hand, an endogenous game weaves educational content into the game play so that students can directly learn while exploring or performing other interesting tasks. 'Oregon Trail' is one of the most iconic commercial endogenous games. The game takes the

learner on an adventure across the United States in a covered wagon. Players attempt to keep their companions alive by rationing supplies, and making other decisions.

Dovan Rai is developing a serious game at Worcester Polytechnic Institute called Mily's World. The game teaches math by engaging children in a world inhabited by a nice young girl named Mily. The student solves math problems, and by doing so earns various rewards (e.g. a new puppy for Mily).

Mily's World provides a laid back atmosphere in which students participate without competition or pressure. This is meant to ensure that students focus on the problems while enjoying the storytelling and character elements of the game. Also, by providing such a relaxed atmosphere, students can learn in a setting that is unlike the common classroom, where pressure to succeed is high [75]. Mily's World contains a number of intelligent tutoring features that make it a user-friendly way to work on math problems. The game provides increasingly sophisticated hints, and supports students to work through a curriculum at a reasonable pace.

Additionally, the iSTART tutor teaches reading skills by inviting students to read and self-explain texts [29]. The design is based on self-explanation reading training, which research has shown to help students learn reading comprehension. The tutor uses technology to lead students through a curriculum of reading drills, resulting in learning games. MiBoard is an extension of the iSTART tutor that adds game mechanics [29]. MiBoard offers students a chance to progress through the iSTART curriculum in the form

of a board game. Players are each given a token on a large board, and take turns practicing self-explanation of texts. While one student works on a piece of text, the other players spend their time determining the strategies of the first student. This supports all players to stay involved in the game, and also supports for additional learning by providing peer observation opportunities.

Figure 2.6: Example Screen shot of Grockit.com

One particularly nice addition to MiBoard is the ability to earn points and purchase rewards. Although many games include the idea of a score, MiBoard attempts to motivate students by allowing them to use their success to purchase advantages in the game. For example, a player might spend half of his earned points in order to freeze another players piece for one turn. Scoring however is based on each player's own criticisms of another. This requires that students play in an honorable and sociable way.

So far, we have seen two examples of how to add a few simple game mechanics to an intelligent tutor to create a game. The intelligent tutors above translated nicely into a

game. For some systems however, it is not obvious how to create a game. Next we look at Grockit, a system that attempts to make a game out of standardized testing practice.

Students spend hours studying for standardized tests such as the SAT and the GMAT. Grockit is a website designed to incorporate group study sessions and learning in various subjects for students who want to improve their standardized test scores. The site's core functionality contains tools for studying lessons and practicing exams. The site also contains collaborative features such as chatting to encourage group participation.

The developers at Grockit Inc. have added game mechanics to their system, in an attempt to make the work more fun. The site now includes points, badges, quests, and performance statistics. All of these are simple digital rewards for good work. However, the site does not, like MiBoard above, contain any tangible result for attaining these abstract rewards [10]. This system is a perfect example of an exogenous game, because game mechanics are placed "on top" of another system. The core functionality of the system still works the same way, but motivation is increased through game features.

We now describe a few games that attempt to teach concepts using an immersion principle. These researchers believe that serious games are most effective when they simulate real and interesting scenarios through which students learn via experiences.

Researchers at Harvard University have created Multi-User Virtual Environments (MUVEs) for learning. The most notable project is called River City [115], and involves

students exploring an immersive 3-dimensional town in a collaborative effort to solve problems. Students form hypotheses regarding the causes of various problems in the town (bad water, sweeping illness, etc.) and collaborate and quickly discover that there is no one correct answer for these issues. They must think critically about how to test their competing hypotheses scientifically.

Researchers at Carnegie Mellon University have built a tutor that teaches students to recognize the distinctions in past tense verb forms in French [44]. Instead of adding somewhat arbitrary game mechanics to the tutor, such as points or certificates, the team built a simulation through which a player immerses themselves into the practical applications of the material [44]. In this game, students take on the role of a journalist who needs to edit French documents. The interface becomes an email inbox through which students receive assignments and feedback from their boss. The player then works to obtain promotions and other rewards.

This game is interesting because it chooses not to use many normal game mechanics, and instead uses pure simulation. Although being a French journalist may not be the most glamorous application of learning French, it indeed provides students with a practical application for their work, giving their work purpose. Studies indicate that students reported being more engaged and motivated when playing the game as opposed to the tutor alone. However, students playing the game did not have increased learning over their non-game playing counterparts.

Although we described this game as an immersive simulation, it does not fully mask the tutoring elements, and thus is not the most immersive game studied for this review. The next two games we discuss can be described as more complete simulations. Each of these games places students in a world facing a particular challenge. Material is not explicitly taught in these games. Rather, learning is a byproduct of exploring and engaging in the world, and solving its many problems.

Our first example of a fully immersive serious game is Crystal Island. Students take on the role of a member of a science team who is doing research on a mostly uninhabited island. A disease breaks out on the island, causing severe sickness among the members, and the player must then investigate to uncover the mystery of this deadly illness [77].

Figure 2.7: Students explore a 3D environment and investigate a breakout in Crystal Island

Crystal Island features a fully 3D interface and environment in which players explore, and collect important information by meeting and conversing with other agents (figure 2.7). Players can collect other information through realistic events such as reading newspapers or examining objects in the environment. Results of studies with Crystal

Island reflect the struggle to find significant learning gains across gameplay groups. However, results have shown that when data is collapsed, science content learning and self-efficacy significantly increased [119].

Crystal Island has also been used as a platform for investigating ways in which games can learn from players. In particular, studies have shown that the application of Markov Logic Networks can lead to an accurate system for automatically detecting player goals [118]. Although this is not the learning of explicit domain level knowledge, it provides evidence of the promise of leveraging game players for machine learning purposes.

Another simulation game is UrbanSim, designed for military personnel who must learn unique leadership skills that are often in great contrast to anything their education has taught them previously (figure 2.8). UrbanSim is a serious game that simulates hostile situations in foreign countries for military leaders [62]. The simulation takes the form of a turn-based game in which the player experiences simulated battles based on stories from real military personnel. Due to the turn based style of play, each decision made by a player can be projected through several different AI components.

An intelligent tutoring system uses various models to judge the decision and provide necessary feedback. Also, another AI component simulates the mental attitudes of various agents within the game, and computes how those agents will react to a given situation. In this way, a player must learn how to judge the reactions of those agents. For example, a player might have to adjust to cultural differences when dealing with foreign

agents. Since the behavioral engine can model these interactions, the player is able to learn from experience.

In addition to this, player actions are pushed through a story engine. This engine helps drive the story in accordance with the player's actions in a way that maintains continuity. The story events are mainly derived from actual events recounted by military personnel, and are important for the complete game experience.

UrbanSim is an example of a game that uses artificial intelligence techniques to model realistic situations. As a result, players can be immersed in a realistic simulation of a complex environment, and learn important techniques through interactions with simulated agents. Using artificial intelligence to increase the simulation fidelity of games seems to be an opportunity for games to make large increases in student motivation and learning.

Figure 2.8: Screenshots of UrbanSim. The player looks over and analyzes the terrain within which a mission was introduced (left) and designs non-violent strategies for solving problems (right).

Other tutors also attempt to tackle the problem of increasing cultural awareness, namely the Virtual Cultural Awareness Tutor (VCAT) and a serious game called Tactical Dari

[116]. Tactical Dari provides players the opportunities to practice cultural skills in an immersive simulation. Players must make choices within the context of a given scenario, and face consequences dependent on how culturally competent those choices appear to be.

Another prevalent serious game is called Immune Attack [51]. In the game, students run around a virtual body as a molecule sized creature and learn about how the immune system helps keep our bodies safe. The game's most attractive feature is the 3-dimensional exploration of the inside of a patient's body. Students learn about human anatomy, and the molecular detection of diseases and how they can be combated. Immune Attack contains uninterrupted game play, and has a helpful resource called MyLA that allows students to ask questions, view interactive images, etc. to help them solve problems.

The literature often brings up the issue of how to integrate assessment of knowledge when using serious games. Many argue that it is easy to track student actions within a game and derive assessment results from their usage of the system. This is clearly not a great technological challenge, but it may have detrimental effects, specifically that by integrating explicit (versus stealth) assessment with games, the motivation increase that may have resulted from using a game in the first place is inherently lost. Either way, it is important that future research specifically address this issue, so that a set of standardized assessment methods can be established.

Brian Nelson describes one problem in learning assessment to be the heavy usage of specific questions to determine student understanding. Modern assessments focus too much on the memorization of knowledge, instead of the full understanding of issues related to that knowledge [69]. Nelson suggests that games should incorporate assessment by including, for example, quests which can only be completed by understanding a concept. In this way, students demonstrate their understanding of a topic while experiencing possible applications of concepts. This, of course, may become an extreme implementation burden for some concepts.

There exist empirical reports to consider with regard to the use of games and simulations in education, and their effectiveness. One article [76] reviewed 28 studies of serious games. The overall finding was that students did not necessarily learn more when using games and simulations over traditional classrooms, but evidence that learning increases does exist. The survey also reports that additional improvements in fine motor skills, and attitudinal changes do seem possible via games. In spite of this though, evidence also exists that not all game features lead to noticeable student improvements.

Other serious game benefits are being discovered. For example, one study, based on the ARCS model (Attention, Relevance, Confidence, and Satisfaction), correlated motivation, brain activity and skin conductance with serious game play [30]. Researchers used a game called Food Force with 33 individuals (11 females), who played for 45 minutes and answered a questionnaire. All students showed significant knowledge improvement, along with a significant increase in attention and confidence.

2.3 Authoring Tools

The third field we describe is authoring tools, or various approaches for the creation of intelligent tutoring systems by non-programmers. We briefly discuss manual authoring tools, which provide non-programmers with effective tools for designing tutoring systems (or sub-components of tutoring systems). Our approach is dramatically different from that used by researchers who employ traditional authoring tools, and so our focus in this section is on the relatively few attempts to automatically construct tutors computationally.

2.3.1 Tools for the Manual Creation of Tutors

Although authoring tools that provide services for the manual creation of tutors is not our focus, we must note the extensive work in providing tools that assist non-programmers in constructed intelligent tutoring systems. Most relevant to our work are tools that assist with the creation of expert knowledge. One such system is 'Demonstr8' [14]. Demonstr8 is a prototype tool that explores authoring domain content by having experts provide examples. The system then attempts to generalize these examples into rules that can be applied to a variety of cases. Other tools for authoring domain content include D3 Trainer and Training Express [66].

Many other authoring tools exist that support the creation of simulations (DiAG, RIDES, MITT-Writer, ICAT, SimQuest), curriculum sequencing and planning (DOCENT, IDE,

Expert CML), and intelligent systems (CALET, GETMAS, Interbook). More detail on the specifics of these manual authoring tools can be found in [67].

Additionally, authoring tools have also emerged that allow non-programmers to quickly develop serious games. The most notable of these systems is called Thinking Worlds [94]. Thinking Worlds is an authoring tool designed for users who wish to quickly build and distribute 3-dimensional simulations and games. In addition, Thinking Worlds contains features that are specifically useful for educational applications. Games created with Thinking Worlds include a band management game [94] and a Human Sensitivity Game [94]. The obvious drawback to any authoring tool, but especially to ones like these that utilize 3D graphics, is the limitation in flexibility that can be obtained by a non-programmer. Of course, Thinking Worlds provides the functionality necessary to extend the provided functionality, but doing so requires programming experience that negates the original intent of the authoring tool.

2.3.2 Tools for the Automatic Creation of Tutors

Some authoring tools contain features for the automatic creation for portions of intelligent tutors. There are very few systems, however, that claim to create entire tutors from scratch (except CTAT and ASPIRE, see below). However, as previously stated, portions of tutors can sometimes be automatically generated. In this section, we outline some more prevalent authoring tools with specific mention of automatic generation features.

ASPIRE is an authoring tool for constraint based tutors [65][124][125]. ASPIRE takes authors through several steps of development, including ontology building, modeling problems and solutions, building an interface, etc. One key thing to note is that ASPIRE contains algorithms that attempt to automatically generate constraints (both syntax and semantic constraints) from the given ontology. This system still requires large amounts of expert time for developing the ontology, in addition to numerous problem examples and solutions.

Another approach is to model the student teacher paradigm. Disciple [93] is a system for an expert to use to train an intelligent agent, under the master and apprentice paradigm. The authors claim that knowledge engineering time is greatly reduced by utilizing Disciple, but most of the work is still done by the domain expert. Disciple requires experts and knowledge engineers to create an ontology of historical concepts, etc. and then to provide examples of rules for generating questions. Machine learning algorithms use these examples to train an intelligent agent. Disciple creates generalized and specific rules based on examples, and attempts to learn rule generation. Generated rules can be presented to an expert, who can either reject or accept the rule as being legitimate. Authors use experimentation to show that after the agent is developed, its rule generation is extremely accurate (>95%).

This is similar to the proposed work, but different in a few important ways. Most notably, the authors do not comment on the time commitment for full development of the ontology, plus the training of the intelligent agent. It seems that knowledge engineers

still maintain a large role in development of knowledge, and that experts may require substantial modeling knowledge in order to correctly train the agent. Our approach, rather, is to utilize the work of students for the creation and adaptation of a tutor.

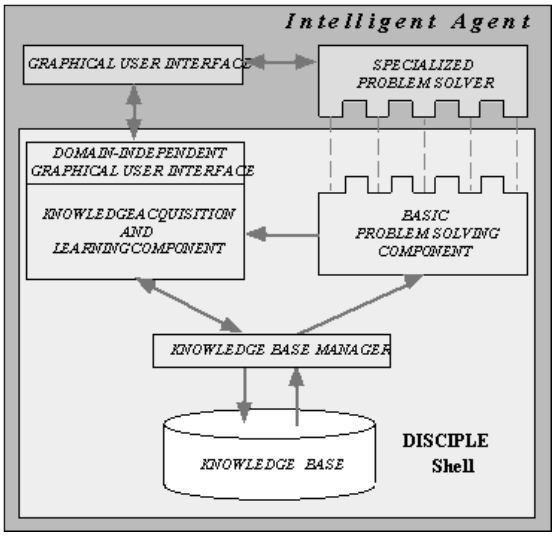


Figure 2.9: Architecture of the DISCIPLE Authoring Shell

The CTAT authoring tool also contains automatic features for using rule-based systems to emulate the cognitive processes of problem solving [32][122][123]. Essentially, this is an effort to defer the need to write production rules, and instead provide example student behavior from which rules might be inferred.

In general, authoring tools are very much geared towards well-defined domains, such as Math and Physics. The CTAT authoring tool allows authors to carefully define a domain by providing examples of correct solutions and student errors. The authors claim a decreased development ratio of 25 hours of development time for 1 hour of student tutoring, down from the previous 200:1 ratio.

Additional research presents the idea of expert knowledge bases and describes efforts in deciding whether or not the knowledge is complete [9]. These researchers present a tool in which an expert works on an EKB, and a system helps authors focus their effort on completing crucial parts of the knowledge base. This tool sounds somewhat similar to the knowledge refinement tool presented in our work. However, such research has limitations. It takes a long time for an expert to build the entire knowledge base from scratch, and thus our approach of having students create knowledge may be more efficient. Utilizing a similar approach for recognizing incomplete sections of the knowledge may be applicable to our research.

2.4 Crowdsourcing

The fourth and final field we describe is crowdsourcing, in which the "crowd" is utilized for the purpose of breaking up large jobs into smaller manageable chunks, and then accomplished by a plethora of anonymous individuals. This field has garnered much research interest, in part because the jobs/tasks can include anything from annotating or translating text, to solving cutting edge math problems, or verifying language translations. The "crowd" is generally made up of individuals willing to contribute to the needed task, either voluntarily or at a very low pay rate. Our work intends to utilize a similar approach in which students contribute to the development of an expert knowledge base by playing a serious game. Thus, the work in crowdsourcing is of interest to us. We begin by summarizing methods for motivating the crowd, and then summarize a number of specific crowdsourcing projects, and close this section by distinguishing our work from traditional crowdsourcing research.

2.4.1 Motivating the Crowd

One of the more pressing issues in the field of crowdsourcing is motivating individuals to contribute to the task at hand. These systems generally do not require a litmus test for participants, but rather work to find ways to maximize the number of contributors. The three premier approaches for motivating the crowd are as follows:

> ***No Motivation*** – Many crowdsourcing projects do not address the need to motivate the crowd. The task at hand is presented at face value, and participants are asked to contribute.
>
> ***Financial*** – Financial motivation is probably the most prevalent motivation strategy, in which small financial incentives are offered for every individual contribution. This financial gain, which is often very small, can accumulate to more substantial numbers as participants increase their contribution. The most notable system utilizing this approach is Amazon's Mechanical Turk [5].
>
> ***Fun*** – A more recent emergent strategy is to make the tasks fun. This involves masking the task as a game or other form of entertainment. This includes Google's Image Labeler [42] which we discuss later in this section.

With these in mind, we now turn our attention to some practical examples of crowdsourcing projects.

2.4.2 Examples of Crowdsourcing

We begin by discussing Mechanical Turk [5], an Amazon project that allows users to collect data from multiple people in the form of small tasks in succession. Mechanical Turk offers great flexibility in the nature of the task that is requested, but it is often necessary for programmers to set up such tasks. Additionally, task creators can choose to force contributors to prove their qualifications if desired.

One particular experiment of crowdsourcing focused on collecting user feedback on the quality of Wikipedia articles [53]. This is an example of what is called a "micro-task markets". The researchers found that immediately verifiable tasks are much more useful than vague questions (How many images were in the article? vs. Were there enough images in the article?). They also note that it is important to make doing the task correctly as easy and painless as possible to deter users from maliciously "gaming" (a term that describes users who try not to learn, or accomplish a task in the way intended, but rather try to gain an advantage by leveraging some aspect of the task design).

We propose that one design that could greatly increase honesty in answers is having a game that challenges players to (potentially unknowingly) contribute information to some cause. Mechanical Turk pays users to contribute information in a survey like way. But a game, integrated with an intelligent tutor, can challenge students to create knowledge in a way that does not feel like work. This altered incentive could lead to increased honesty.

Crowdsourcing for annotation purposes are primarily used with the three motivators as described above. Some research has focused on crowdsourcing and how applications can utilize it [102]. However, our approach is unique in that we are utilizing a group's effort that requires little extra motivation. School students required to use a tutoring system will do so, and the quality of the tutor may increase the quality of the results, but the participation will be available regardless.

Another example is PlateMate, a crowdsourcing system that attempts to utilize Mechanical Turk workers to identify the nutrition facts of various meals [70]. Users can take a picture of their food, and upload the photo. The task is split into multiple micro-tasks, including the tagging, identification, and nutrition calculation of foods.

The researchers found that their system was able to approach the accuracy of expert dieticians, but in a more scalable way. However, they had to deal with issues such as "Lazy Turkers", who don't put enough effort into their respective tasks, and ambiguous photos (fat free latte vs. coffee with cream).

Another interesting piece of work involves a virtual helpdesk for a university course [43]. This software utilizes student models (created through university students' use) to understand the knowledge level of individual users (backed by a two-level expert knowledge base). This information is updated using a variety of direct and indirect sources of evidence (tests, relations of knowledge, etc…). When students need help, the system looks for resources to help them. The resources include other students, and thus

the system tries to match students who need help with others who are able to provide it. The system is very generalized and can be used for any domain.

The researchers have also migrated their system to a multi agent system (MAS) architecture in which each student has their own 'Agent' [100]. In addition there are many universal agents. These agents communicate with one another (negotiate) to determine how interactions and potential help can be provided. These agents attempt to maximize the gain for their student.

I believe that this work is strong in that it views peers as potential resources for knowledge. It seems to be dependent, however, on having a relatively large and active community. In particular, this is a crowdsourcing approach in which the "crowd" is not necessarily cognizant of the fact that they are contributing to the task.

2.4.3 Games with a Purpose (GWAPS)

GWAPs are games that produce, as a side effect of human game play, helpful computational tasks (e.g., train an AI algorithm) [99]. For example, the Input-Output ESP game gives two players an identical image. Anonymously, they are both invited to create a description of the image and earn more points if their descriptions are similar. In this way, the description is guaranteed to accurately describe the photo, thus training image AI algorithms.

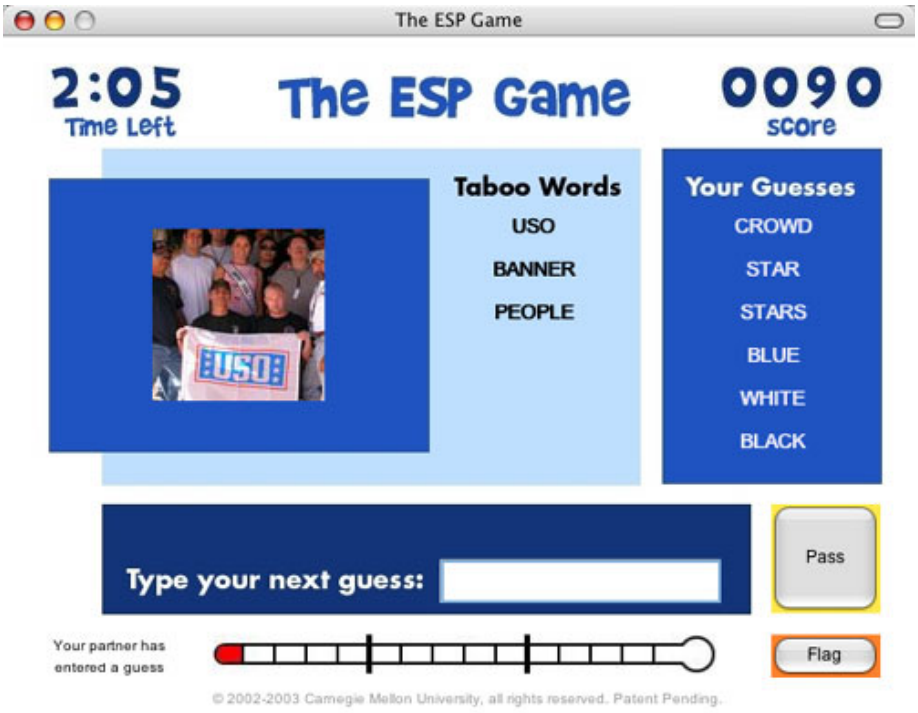


Figure 2.10: In the ESP game, each player describes the image shown (left) in an attempt to match an anonymous partner's description.

A very similar game to this one is Google's Image Labeler, which invites players to label various images, in an effort to improve image search results.

These games incorporate clever designs that support users to be as accurate as possible. In particular, "teammates" within the image labeling game are anonymous, and thus have no connections to one another except for the image. Thus, because the goal of the game is to input the same answer as your partner, it can be safely assumed that accurate input matches are those that occur because both parties describe the photo. The only notable way to circumvent this strategy is prior collusion. However, if the partner anonymity is implemented carefully, this should be impossible.

Variations on this game design have appeared that take advantage these same strategies. In particular, other games that have arisen are as follows:

> ***Inversion-problem games*:** one player receives input and describes it. Then, other player must use the description and attempt to guess the original input. This type of game can be effective for getting people to provide accurate descriptions of images or to translate sentences.
>
> ***Input agreement games*:** both players obtain (possibly different) input and describe them individually. Then, the players are able to view each other's descriptions, and must agree whether or not they were originally given the same input.

Other game mechanics (levels, points, etc.) are included to increase player motivation. This can sometimes be a tricky design decision, as the goal is to motivate players to work quickly and efficiently. Timers are often used to increase the game's sense of urgency.

There are many other examples of game and task variations in which players "unknowingly" contribute to complex computational tasks. Tag a tune [39] is a game in which anonymous players listen to a song and describe it. They then view each other's descriptions and must determine if they are listening to the same song.

Squigl [39] is an interesting game that supports computer vision applications. An image appears on the players screen along with a single word. The player is asked to use his or her mouse to trace the part of the image that is being described. For example, if the word is cat, then the players must find the cat in the image and trace around the cat exactly.

Figure 2.11: A player attempts to outline a doll in the image provided by the game Squigl.

Most of these games focus on the need for machine learning applications to use copious databases of correctly annotated data for training. Players of these games are essentially providing the training sets for such algorithms. Other applications are possible however. Phrase Detectives [17] is a game that challenges language and literature lovers to describe semantic relationships between pairs of English words. The goal is to have players create a semantic web of linguistic information, usable by computational linguistic algorithms. This project shows the promise of not just labeling data, but establishing semantic relationships between data as well.

Another game that supports semantic relationships is "Flip It" [39]. In this game, players are given a set of face down cards. Players take turns flipping pairs in an attempt to find matches. The caveat is that exact matches don't exist. Players are instead challenged to

find matches with similar qualities. This serves to train algorithms that must establish subjective similarities in the features of images.

Games with a purpose can be created to benefit other disciplines other than Artificial Intelligence. For example, Foldit [21] is an online puzzle game about protein folding. Players are presented with a structure, and must fold it to match given proteins. Researchers closely analyze the best scores in an effort to determine whether or not there is a native structural configuration that can be applied to the relevant proteins [21] in the "real world". Players are thus contributing to the scientist's ability to create solutions to "real-world" problems.

Similar games have also emerged that attempt to build deep ontologies by analyzing player performance. For example, OntoPronto [84] is a quiz game for vocabulary building. In OntoPronto, two users read a random Wikipedia article and are rewarded points and may proceed if they can agree on the ontological placement of the article's concepts. If not, they receive another random article. Simply counting the majority opinion among players validates the evolving ontology.

Other games with this similar purpose exist, including Virtual Pet Game [105], which challenges students to contribute to common knowledge ontologies by simulating the care of a virtual pet. Interactions with the pet are actually communicated to other players, who provide responses. Other games of this nature include 'Rapport Game' [105] and 'Guess What!?' [95].

2.5 Conclusions and Problem Statement

This chapter summarizes current research from a variety of fields. Our goal was to present existing research in a way that makes clear the potential overlaps with our proposed research and how they support one another in ways that lead to interesting possibilities. We began by summarizing pedagogy for ill-defined domains, with specific respect to its relevance to the Rashi system (see Chapter 3) used for the experiments in this dissertation. We then discussed intelligent tutoring systems that utilize advanced artificial intelligence techniques to provide unique individualized support to students learning a new subject. In particular, intelligent tutors have been shown to significantly increase learning (compared to typical classroom approaches). We also note the vast research in the field of serious games. These systems are not necessarily intelligent tutors but attempt to provoke more entertaining, immersive learning experiences by incorporating game mechanics. Studies in this field have been hard pressed to show significant learning gains with games as compared to typical intelligent tutors (the gains are approximately equal); however studies that provide evidence of learning do exist [121]. Additionally, we have seen that many studies show an increase in student immersion, motivation, and interest.

We believe that a well-designed serious game is capable of motivating and encouraging a consistently high level of work from students. For this reason, we believe that intelligent tutors and serious games require methods that take advantage of the plethora of information being submitted by students while using a system. We note, in particular, the

benefit of a tutor that uses an expert knowledge base to reason and provide assistance to students. We detail some of the literature on expert systems in this chapter, and note that the biggest drawback of such systems is the tedious development time, and need for an expert in a particular field.

We thus attempt, in this project, to develop methods to greatly decrease the development time of expert knowledge bases for use within intelligent tutors and serious games. Our approach is to consider a development cycle in which a system without an expert knowledge base is created first. Then, the work of students within this system is coalesced to construct an expert knowledge base automatically. We propose algorithmic methods for observing student behavior within an intelligent tutor and constructing expert knowledge. We hypothesize that this approach is not only feasible, but also efficient. We also propose that a serious game is the ideal venue for obtaining the highest quality work from motivated students.

To this end, we note the work on authoring tools for intelligent tutors, despite the fact that our proposed approach vastly differs. Our approach, instead, adopts ideas from crowdsourcing, in which a plethora of anonymous (possibly non-expert) users are asked to help solve a problem. In particular, we note the work on games with a purpose (GWAPS) in which the crowd is used to annotate large datasets by playing addictive games.

It is our hope that our approach excites designers of intelligent tutors and serious games to leverage the knowledge of their users. In particular, our approach considers the fact that even though the serious game is offering educational services to the player, the player can still offer knowledge-building services back to the game. In this way, a system is constantly learning from its students and is, in turn, becoming more effective at providing dynamic feedback to those students. In addition, it is our hope to provide evidence that this approach vastly decreases the necessary development time of intelligent tutoring systems and serious games.

CHAPTER 3

PROGRAM AND ALGORITHM DESIGN

In this chapter we discuss, in detail, the features and algorithms that were developed in order to investigate our ability to automatically infer expert knowledge from student interactions within an intelligent tutor. In particular, we developed an intelligent tutor with a fun and interactive gaming feel. We then developed algorithms that study users' behavior within this system, and infer expert knowledge for the system from student answers.

This section begins by describing Rashi, an intelligent inquiry tutoring system that has been developed at the University of Massachusetts, Amherst. We then describe some game mechanics that have been developed as additions to the Rashi system.

We next define the knowledge acquisition algorithm in detail. This algorithm accepts student usage data from Rashi as input, and outputs an Evolving Expert Knowledge Base (EEKB), which is also defined in detail in this chapter.

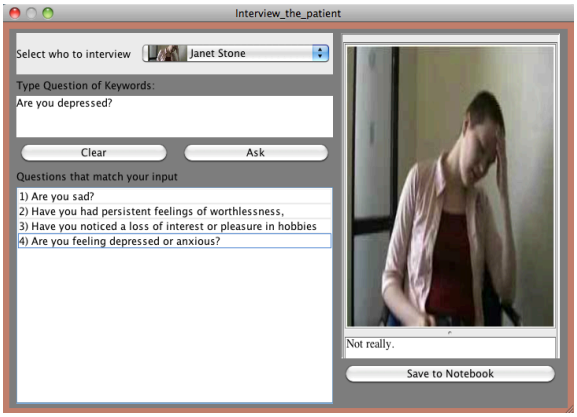
We acknowledge that the algorithm may very well be erroneous. Thus, we also define an independent application called 'Dr. Doctor', which is an instance of a knowledge-refinement game (KRG)[4]. This game is intended to be used by a small group of experts, and is designed in a straightforward manner to quickly refine the expert knowledge base that is created by the algorithm.

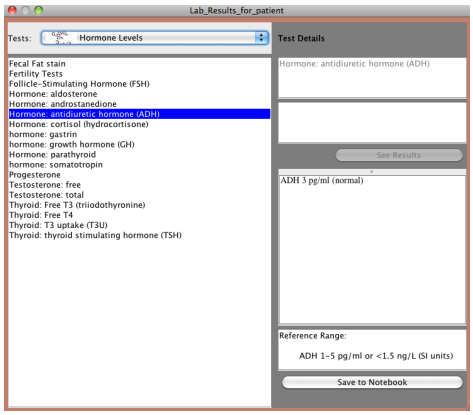
[4] *We use the term Dr. Doctor to refer to our specific instance of a KRG and use the term KRG generally to define a game that is used by domain experts to refine the evolving expert knowledge base (EEKB).*

3.1 Rashi: The Intelligent Inquiry Tutoring System

Rashi is an existing well-vetted inquiry learning system that provides tools and environments for students to engage in authentic learning experiences by considering real-world problems [33, 34, 35]. The system provides case descriptions for students to investigate, along with information about how to approach each problem [33]. Rashi is domain independent and applications in several domains have been created and tested. This research focuses primarily on the *Human Biology* domain.

Various data collection vehicles (interactive images, interview interfaces, video and dynamic maps) provide open-ended spaces for student exploration and to acquaint students with methods commonly used by professionals to access and organize information in their field. In the *Human Biology Tutor* (the domain used in the current research), students evaluate patients and generate hypotheses about their medical condition. Patients' complaints form an initial set of data from which students begin the diagnostic process; for example, virtual patients are interviewed about their symptoms (Figure 3.1). Data is made visible by student actions (e.g., asking for chest x-rays, performing a physical examination of the patient, or running lab tests (Figure 3.2)). Students move opportunistically from one inquiry phase to another as they sort, filter, and categorize data in order to form hypotheses about the patient's illness [104].



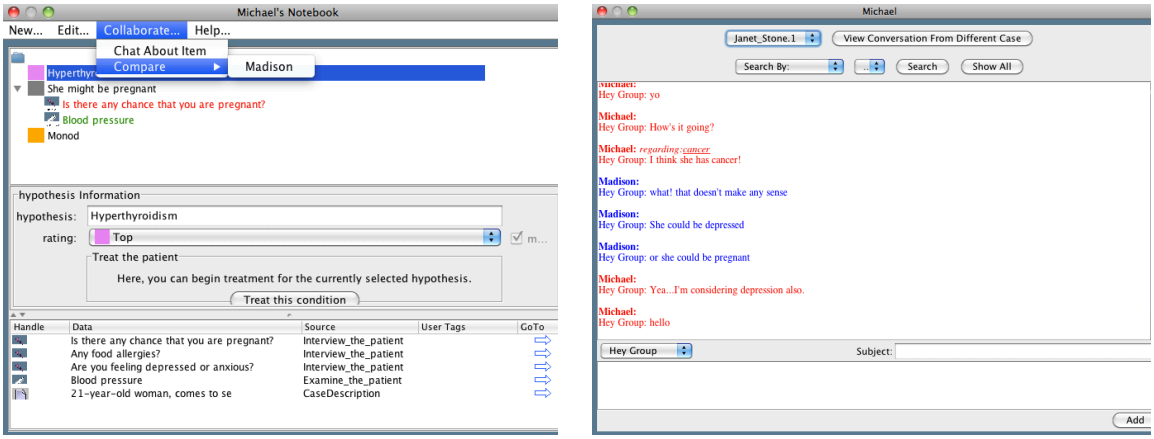


Figures 3.1-3.2: Students interact with Rashi through several interfaces, e.g., they might collect data by interviewing the patient (left) or running various lab tests (right).

The current Rashi system makes use of an expert knowledge base (EKB) for any case in which a student is working. The EKB is a directed acyclic graph that represents pieces of knowledge relevant to the case, as well as the interactions and relationships between this knowledge. Because this EKB is a representation of the knowledge relevant to the case on which students are working, it can be used as an engine for providing students with relevant feedback and coaching. Within Rashi, students have access to a coach. The tutor judges student actions, and compares their actions against the expert knowledge base to determine the location within the expert knowledge where students are currently working. The coach then finds relevant information in the nearby knowledge space to offer feedback to the student. In addition, the coach compares a student's actions and arguments against the work of other students in the group to determine when collaborative activity is optimal [35].

The Rashi system uses collaborative capabilities to enable students to view and share work within a group. These tools support students allowing them to view each other's notebooks (Figure 3.3) and to drag and drop both data and hypotheses from others'

notebooks to their own. This supports a variety collaborative activities ranging from students working in tightly knit groups, where each student takes on a role and contributes in a specific manner, to students working mostly independently but sharing ideas and thoughts when reaching an impasse. The system also provides a chat facility that enables students to discuss issues with members of their group (Figure 3.4). Several features, including text coloring, filtering, and new message notifications increase the usability and quality of the discussion tool.



Figures 3.3-3.4: Rashi provides a notebook (left) to support students to organize their work and to collaborate with their team members. Students can also collaborate with team members by using our free-text chat room (right).

Students can create a subject for each message, which allows the team to focus on a specific topic (Figure 3.4). Chat messages can be filtered by these topics and students can easily respond to the subject by clicking on it. In addition, Rashi supports users to use a one-click method of automatically setting the subject of a new conversation to the contents of an existing Rashi notebook item. This creates an internal link between the conversation and the notebook item, allowing a confused group member to click on the chat subject and be quickly taken to related work in a group member's notebook.

Rashi is implemented using Java and is architected as a standard client server model. The server stores all relevant information regarding the cases and sends large XML messages to the client to communicate this knowledge. Student data is stored using a MySQL database.

Past research has investigated the impact of the aforementioned coaching and collaboration features on student learning. Specifically, it was observed that collaboration features lead to an increase in positive student behavior [117]. The same effect was observed without significance for automated coaching, mostly due to the low occurrence of students requesting such help. There was evidence however that once requested, coaching did lead to improved student work [117]. The same study illuminated the importance of a strict pedagogical approach when incorporating intelligent tutors for inquiry learning. When carefully designing a workflow and engaging teachers in that workflow, the observed quality of student work rose dramatically. Thus, there is sufficient evidence that the work described in this dissertation is important, as a developed knowledge base is required to fully support these teaching efforts.

3.2 Adding Game Features to Rashi

Although the conditions necessary to classify an application as a "game" rather than just a "tutoring system" are not clearly defined, we argue that the Rashi Intelligent Tutoring System is in fact a game. Rashi encompasses many game features, including fantasy

(playing doctor), open-ended exploration, puzzle, and intrigue. However, there are a number of common game mechanics that Rashi noticeably lacks.

In order to enhance the Rashi system, we implemented an additional patient treatment feature, which supports students to actively monitor a patient's status. A patient's health will slowly decrease over time, and students are invited to administer treatments for various conditions and observe how the patient's health changes as a result of these actions.

3.2.1 The Patient Status Panel

Our first additional Rashi feature is a patient status panel, located within the main Rashi window, that simply provides students with an easy way to monitor, in real time, the condition of the patient (see figure 3.5). This panel consists of three parts:

> ***Patient Character:*** An animated character provides a visual representation of the patient, including a few emotions that are loosely correlated with his or her health. Healthy patients look happy, while worsening patients look ill. Additionally, the administration of treatment is visually animated for students.
>
> ***Health Bar:*** A health bar displays a quantifiable view of the patient's health. The bar is color coded to display healthy (green), sick (yellow), or critical (red) conditions. This health bar is dynamically updated depending on the patient's current condition and any currently applied treatment.

Treatment: A panel displays a text representation of the condition for which proper treatment is currently being administered. Thus, once students set the treatment for the patient, they can observe both the treatment and its effects in unison. If no treatment is selected, then the health of the patient slowly decreases, leading to a sense of urgency for the student.

Figure 3.5 shows a prototype of this patient status panel. The panel is conveniently located within the main Rashi window, making the monitoring of a patient convenient for students.

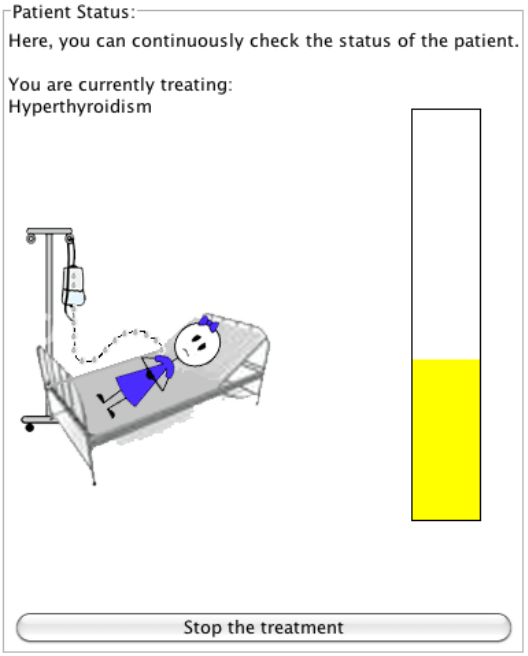


Figure 3.5: The patient status panel showing a representation of the patient as well as a health bar.

3.2.2 The Treatment Button

The treatment button supports students to set a condition for which the patient should be treated. Within the Rashi notebook, students select a hypothesis and then select the condition to be treated. This menu is shown in figure 3.6.

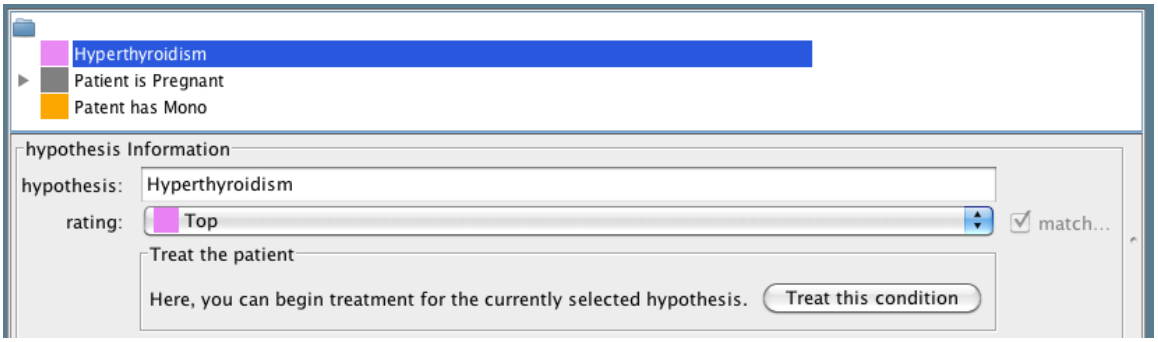


Figure 3.6: Students can select a hypothesis and treat the patient for that illness within their notebook.

Upon pressing this button, the patient status panel is immediately updated to reflect that the patient is being treated for the condition, and the health of the patient begins to be affected. For this initial prototype, students are not required to input the specific method of treatment (they are, however, encouraged by the system to look up this information). The system assumes that the condition is always treated accurately. Future versions of this feature will allow students to define the specific treatment.

3.2.3 Patient Reactions to Treatment

In an attempt to provide students with "reward", the treatment they select directly affects the "health" of the patient, reflected by the patient status panel. Rashi analyzes the selected treatment, and infers how the patient might be affected by the application of that treatment.

For example, if the student provides an accurate diagnosis and treatment, the health of the patient will slowly increase over time, reassuring students that their hard work is paying off. Likewise, an inaccurate diagnosis and incorrect treatment may not affect the

patient's health at all, or worse, may actually cause the patient's health to deteriorate. However, we hope not to discourage our players, and thus no diagnosis will result in terminating the patient.

Due to the immediate needs of these features, they are currently hard-coded for a single Rashi case. In the future, this diagnostic information will be incorporated into the evolving expert knowledge base.

3.3 Algorithm for Knowledge Acquisition

In this section, we define in detail the algorithm for analyzing student behavior within a serious game and inferring student knowledge in turn. We begin by defining an evolving expert knowledge base (EEKB) in detail along with actions that can be performed on an EEKB, providing pre / post conditions for each action. We also include pseudo-code for those actions where relevant.

3.3.1 Defining the EEKB

First, we define a general evolving expert knowledge base in formal nomenclature. An EEKB, in a particular domain D, is defined as a set of nodes and relationships, such that:

EEKB(D) = (N, R) where:
N is a set of nodes
R is a set of relationships
D is the target domain (e.g. Biology, Law, etc.)

We now define nodes followed by relationships between nodes.

Node = (desc, type, conf) where:

desc is a string representation of the description of this node

type ε NodeTypes is the type of node (e.g. evidence, data, etc…)

0 ≤ conf ≤ 100 is the confidence that Node is true for domain D

Relationship = (n1, n2, type, conf) where:

n1, n2 ε N ∧ n1 ≠ n2

type ε RelTypes is a string defining the relationship from n1 to n2

0 ≤ conf ≤ 100 is the confidence that 'value' from n1 to n2 is true

NodeTypes and RelTypes are defined as:

NodeTypes(D) = {type | type is a string; a type of node representable in domain D}

RelTypes(D) = {type | type is a string; a possible relationship between nodes in D}

For example, NodeTypes might include such categories as hypotheses (e.g., "Patient has hyperthyroidism") and data (e.g., "blood pressure is high"). RelTypes might include categories such as 'supports' (e.g., "high blood iron count SUPPORTS hyperthyroidism"), 'strongly supports', 'refutes', and 'strongly refutes'.

3.3.2 Defining Actions on the EEKB

In this section, we outline the actions available within an EEKB. Our knowledge construction algorithm directly utilizes these actions, which for the most part, are straightforward. We define them here, along with clarifying pseudo-code, pre-conditions, and post-conditions.

CreateNode (n, type)

Pre-Condition: ∀ node ε N: node.desc ≠ n.desc ∧ type ε NodeTypes

If a node with description 'n' does not exist in EEKB

Create a node with description n, type, default confidence 10%

Add this new node to the EEKB

Else ignore node creation request

Post-Condition: N ← [N ∪ {n, type, 10}]

EstablishRelationship (n1, n2, type)

Pre-Condition: (∀ rel ε R: !(rel.n1 = n1 ∧ rel.n2 = n2) ∧ type ε RelTypes

If relationship between 'n1' and 'n2' does not exist in EEKB

Create a relationship with n1, n2, type, and 10% confidence

Add this new relationship to the EEKB

Else ignore creation request

Post-Condition: R ← [R ∪ {n1, n2, type, 10}]

UpdateConfidence(node, newConf)

Pre-Condition: node ε N ∧ (0 ≤ newConf ≤ 100)

Post-Condition: node.conf ← newConf

UpdateConfidence(rel, newConf)

Pre-Condition: rel ε R ∧ (0 ≤ newConf ≤ 100)

Post-Condition: rel.conf ← newConf

CombineNodes(n1, n2)

Pre-Condition: n1, n2 in N AND n1 ≠ n2

Let rFrom be all relationships stemming from n2

Let rTo be all relationships going to n2

Remove node n2

For all rel in rFrom

Remove rel

Add new edge (n1, rel.n2, rel.type, rel.conf)

For all rel in rTo

Remove rel

Add new edge (rel.n1, n1, rel.type, rel.conf

Post-Conditions:

$N \leftarrow N - n2$

$newEdges \leftarrow \{ (n1, y, type, conf) \mid (n2, y, type, conf) \in R\}$

$\cup \{ (x, n1, type, conf) \mid (x, n2, type, conf) \in R\}$

$R \leftarrow (R - \{rel \in R \mid r.n1 = n2 \text{ or } r.n2 = n2\}) \cup newEdges$

3.3.3 The Algorithm

We now have the tools necessary to define an algorithm for the automatic construction of an evolving expert knowledge base via student work and observation. We begin by discussing the format with which propositions are taken from Rashi. We then describe and provide pseudo-code for the algorithm that builds an Expert Knowledge Base from a list of these propositions. Lastly, we provide an example use case of how the algorithm deals with a small example dataset.

Rashi propositions come exclusively in XML format, and encode their information with various attribute names. The attributes of importance are:

Type: Whether this proposition is a hypothesis or data (both node types), or a relation (edge type).

ID: a unique identification number

shortDescription: Annotated data describing the contents of the node. This is used for node types only.

isSupports (edges only): This provides the annotation for edges in human biology. Does the edge describe a supporting or a refuting relationship?

From (edges only): Node that an edge is coming from.

To (edges only): Node that an edge is going to.

Timestamp: The date and time when this proposition was created.

The algorithm does not require data in this exact format, tailored to the human biology tutor. These specific propositions are handled by an external method that unpacks the XML and determines the most relevant information. The relevant data is then passed into the main algorithm function for nodes or edges (whichever is appropriate), generalized for any domain:

The algorithm contains two handler methods, one that deals with analyzing an incoming node proposition and another that analyzes the same for edge propositions. These methods accept generic data in order to keep the algorithm generalizable, domain independent, and re-usable.

HandleNodeEvidence(nodeData:String)

HandleRelationshipEvidence(fromNodeId:int, toNodeId:int, relData:String)

These methods work in a very similar fashion, the pseudo-code for which is presented below. The currently generated EEKB is searched using a simple text search engine [106]. Although this could cause problems (for example, if two students spell the same condition differently), any issues will be theoretically cleaned up by the knowledge refinement game (described in section 3.4).

```
Current generated EEKB is searched for potential matches to this new node

If the node / edge is not found already present in EEKB
     Then it is added with low default confidence of 10 percent

If it is found:
     The confidence of this information is increased

A record of this addition is added
```

Figure 3.7: Pseudo-code for incorporating additional evidence into EEKB

3.3.4 Example Algorithm Use-Case

In this section, we briefly provide an example use-case of how the algorithm reacts to a small sample of student data. Figure 3.8 shows how the algorithm will react to one student's set of input data. On the left, is the student's data, starting with the hypothesis created and data collected. The bottom half of the data represents relationships, which are defined by the from-node's id number, the to-node's id number, and some text representing the nature of the found relationship. Given just a single student, the algorithm simply constructs the student's argument and there is little chance of any conflicts.

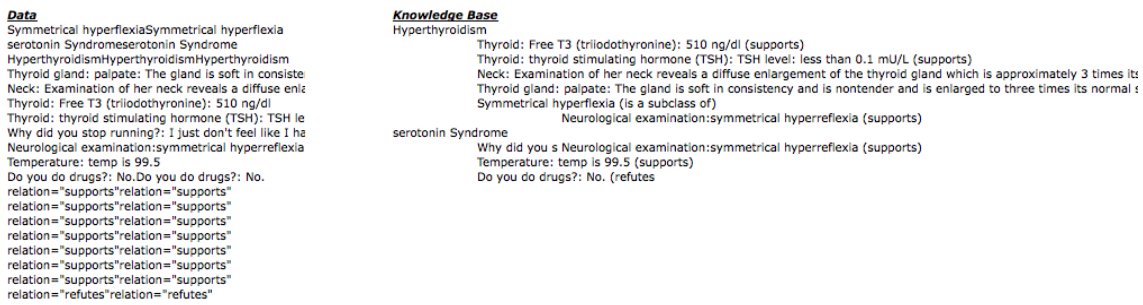


Figure 3.8: Expected knowledge base (right) after algorithmic analysis of one student's data (left)

The first student's argument (figure 3.8 left) is constructed accurately in the knowledge base (figure 3.8 right). However, all confidences (not shown in the figure) default to ten percent after the first student's activities are recorded, meaning that we need other students to verify these assertions before they are accepted as truth. Figure 3.9 below shows how this EEKB will evolve after incorporating a second student's Rashi data. Most notably, some hypotheses are repeated and thus the confidence of these hypotheses is increased (e.g., for hyperthyroidism). Additionally, some of the relationships are also recognized as being repeated, and their confidence is increased.

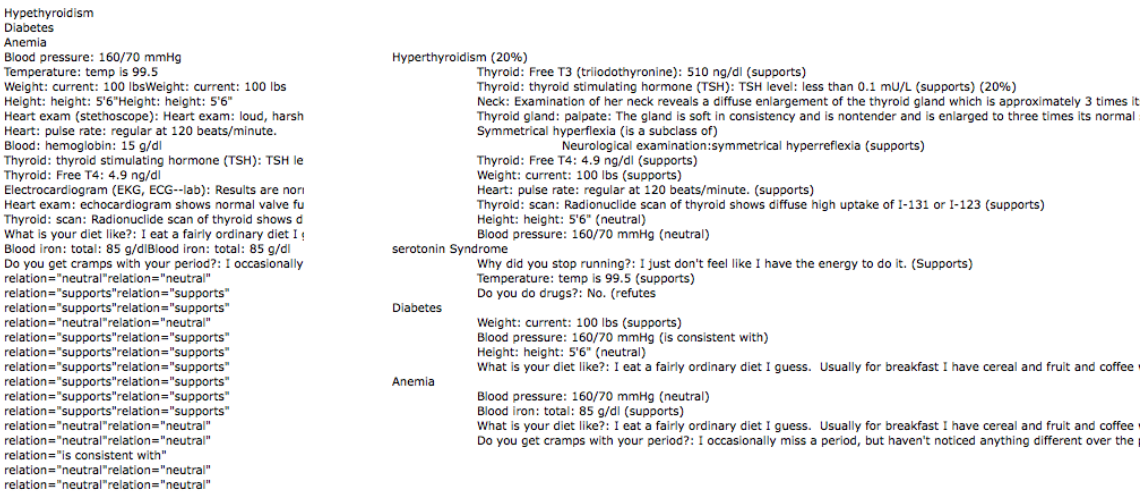


Figure 3.9: Expected knowledge base (right) after algorithmic analysis of a second student's data (left). This analysis is in addition to that presented in figure 3.8.

Most notably, some hypotheses are repeated and thus the confidence of these hypotheses is increased (e.g. hyperthyroidism). Additionally, some of the relationships are also recognized as being repeated, and the confidence is increased.

Although this example fails to show any strong examples of conflict arising between student data, it makes the clear the potential for such anomalies. For example, two students may very well establish the same relationship with different values (e.g. one student says that a piece of data supports a hypothesis while another claims it refutes that hypothesis). Additionally, incorrect spelling would cause the EEKB to contain two separate nodes where only one is necessary. However, this would not be an issue that is irresolvable, as described earlier in this chapter.

Although this example fails to highlight any clear examples of student data that is conflicting, it does highlight a feature of Rashi data that is observable anecdotally; that student data tends to agree more often than it conflicts. This observation makes an approach like the one described in this document more promising.

3.4 Dr. Doctor: A Knowledge Refinement Game

In this section, we present a design for the knowledge refinement game we call "Dr. Doctor". We begin by describing the user experience, showing the game's interface, and describing how players interact with the application. We then continue to describe the infrastructure of the application, along with some details of its inner workings. The KRG

is designed for use by subject matter experts (SMEs) or near experts. By near experts, we mean students who are not less educated than an upper class college student.

3.4.1 Logging in and Playing

Experts (or near experts) are invited to play the game in order to improve the quality of the knowledge base. The expert opens up the knowledge refinement game simply by accessing the game's web page. The experts are presented with a login screen where they either login or register (see figure 3.10). Once the game is loaded, the application searches the existing knowledge base for areas that have low confidence level or need to be refined. Once a suitable section of the knowledge base is found, the information is presented to the expert in the form of a yes/no or true/false question.

To this end, the knowledge refinement game's design has two major non-trivial tasks to perform. The first is question generation, in which a suitable area of improvement within the EEKB is identified and represented as a simple question for a user. The second important task is to update the EEKB. In this phase, an expert's final input must be considered and the knowledge model updated to accurately reflect the truth in the domain.

We present more detail on these phases later in this section, for now, we simply assume those two tasks are handled well, and describe how users interact with the game.

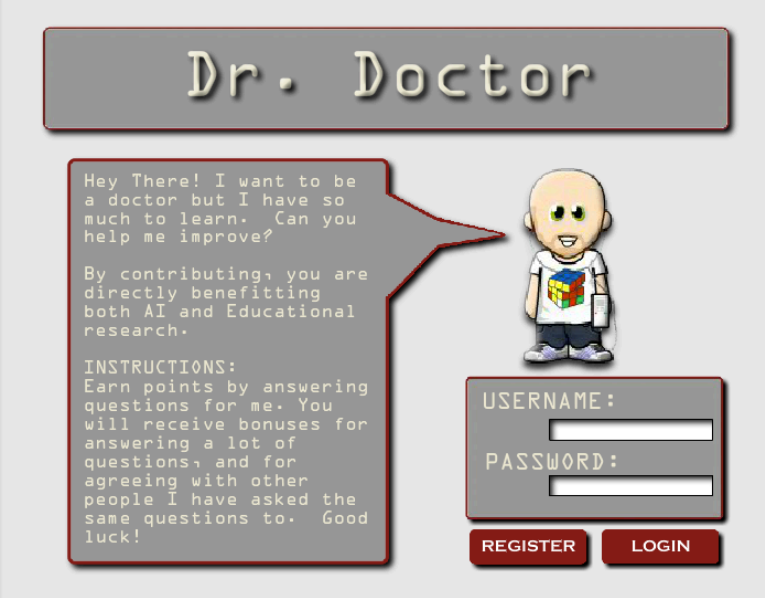


Figure 3.10: Users login to the knowledge refinement game

The expert reviews the information and answers appropriately. Once an answer is ascertained, the application automatically uses the response to update the knowledge base appropriately with increased confidence. This process is repeated for as long as the expert continues to play the game (see figure 3.11).

The game keeps track of a few statistics and displays these in the upper left corner of the screen (figure 3.11). Statistics include the player's contributions to the knowledge base during the current session, the player's total contributions to that knowledge base over all sessions, and the percent confidence of the entire evolving expert knowledge base.

Figure 3.11: Users play Dr. Doctor by answering questions generated about the data in the emerging expert knowledge base, which is built from the automated accumulation of student Rashi input.

In addition, the player is given a score and a level depending on the amount and quality of their contributions. These variables are described in more detail below in section 3.4.4.

3.4.2 Accessing Propositions from the EEKB

The game's first task is to analyze the currently crafted EEKB and search for suitable areas of improvement. The game does this in a series of phases. Phases 1 and 2 are attempts to verify the content currently in the EEKB. After this is completed, the game enters Phase 3 in which new relationships are created between nodes, and similar nodes are combined to streamline the EEKB. A summary of these phases and the types of questions they create is shown below:

Phase 1, Verifying Nodes: Nodes that have a relatively low confidence are retrieved so the expert can confirm whether or not they belong in the EEKB. These questions are of the form "Is this hypothesis valid for this Domain: <Data>?"

Phase 1, Shortening/Clarifying Nodes: Some nodes are detected as having verbose or badly formed data. The expert is asked to offer alternative data, and is encouraged to offer a more pithy response than shown in the current data.

Phase 2, Verifying Relationships: Once the Nodes are verified, the same is done for the relationships between those nodes. Relationships are presented to the expert and responses are reflected in updated confidence values.

Phase 2, Clarifying Relationships: If an expert states that a relationship is incorrect, they are asked to correct the nature of the relationship (if any).

Phase 3, Establishing New Relationships: The game identifies pairs of nodes that may be related to one another and asks the expert whether or not this is the case. The expert is also asked to identify the nature of the relationship, if they've deemed one exists.

Phase 3, Combining Similar Nodes: The game will use String search techniques to estimate whether pairs of nodes reflect the same semantic data. The expert will be presented with such pairs and asked if they are indeed the same topic or idea, and whether they should be combined in the EEKB.

Questions for the experts are generated within each phase at random. However, the potential questions within each phase are enumerated in full before advancing to the following phase.

3.4.3 Reacting to Expert Responses

After asking an expert a particular question and receiving the response, the system must dynamically adjust the knowledge base to reflect this information, on the assumption that an expert response is generally more accurate than that of a student. The game phrases its queries to require mostly yes / no responses, and so the system must only manage the responses from this constrained interaction. In particular, an answer of 'yes' is generally simple to manage because this means that the expert is agreeing that the knowledge base entries are correct, while an answer of 'no' is generally more problematic. There are a couple question types that require String inputs, but they are only presented after a node or relationship has been verified as accurate. These less common text inputs are used merely to help refine the data and make it more pithy and readable.

Table 3.1 below summarizes the question types, responses, and reactions within the knowledge refinement game. We see that for the most part, expert responses directly affect the probabilistic confidence values of the EEKB data. However, there are certain responses that trigger specific question types to follow, while others simply (and quietly) update the EEKB model directly.

Table 3.1 is a slight oversimplification of the KRG's actual execution. Most notably, the game does not update the EEKB model given the opinion of only one expert. This is another area in which a simple probabilistic model is used. Player's responses are stored in the database and are reissued to different experts to confirm the accuracy of the

original response. Because we expect the players to be experts, we consider three responses in agreement sufficient to update the EEKB permanently.

Phase	Question Type	Expert Answer	KRG Response
1	Node Verify	*YES*	Increase Node Confidence, ask followup 'Node Content' question
		NO	Decrease Node Confidence
1	Node Content	*TEXT*	Replace Node data with TEXT
2	Relation Clarify	*YES*	Increase Rel. Confidence
		NO	Decrease Rel. Confidence, ask followup 'Relation Content' question
2	Relation Content	*TEXT*	Replace Rel. Data with TEXT
3	Combine Nodes	*YES*	Combine Nodes into single Node
		NO	Don't ask about these nodes again
3	Establish Relation	*YES*	Create Relation between nodes, ask followup 'Relation Content' question
		NO	Create a "NONE" relation between these nodes

Table 3.1: A summary of the knowledge refinement game's question types, potential expert answers, and the responses provided by the KRG to each potential expert answer.

In addition, an agreement among experts is necessary for motivating players to provide accurate information. In the next section, we discuss the game elements woven into 'Dr. Doctor', and argue that these game mechanics are essential for promoting accurate feedback from experts.

3.4.4 Game Elements

Dr. Doctor, our example instance of a knowledge refinement game, incorporates some simple game mechanics in order to motivate players, and make the experience more enjoyable. In particular, these mechanics are designed to accomplish two goals: 1)

motivate players to continue contributing for extended periods of time and 2) provide incentive for players to provide accurate information.

> ***Feedback Statistics:*** Dr. Doctor displays dynamically changing statistics regarding each player's contributions. This includes the number of all time contributions they've made to the EEKB, the confidence of the EEKB, and the percent increase for which players themselves are responsible.
>
> ***Points:*** Players are awarded points for answering questions to reward players for their contributions and motivate them to continue. This feature is coupled with the level and bonus mechanics described below.
>
> ***Levels:*** Players progress through increasing levels as they garner points. The first level is 'undergrad' and the player is given a higher status (e.g., graduate student, professor) at each increasing level.
>
> ***Contribution Bonus:*** Players are given bonus points for contributing multiple answers in a given session. This attempts to motivate them to continue playing. Players are shown when the next bonus will appear.
>
> ***Agreement Bonus:*** Users are rewarded with a score bonus when their answers are in agreement with that of other online players.

The most relevant motivating mechanics here are the bonuses, which are specifically tailored to encourage players to continue playing for extended periods of time, while simultaneously rewarding them for carefully answering the questions they are given.

In particular, the agreement bonus is used to help ensure users are not so motivated by the other game mechanics that they purposely game the system or provide erroneous answers

to continue playing. This is a lesson from the literature on 'Games with a Purpose' [39]. We attempt to design 'Dr. Doctor' so that the first order optimal strategy (FOO) becomes one in which thoughtful and honest responses are given.

3.4.5 Game Architecture

Dr. Doctor is implemented using a standard client-server model. The client is developed in Flash/Flex 4.1[5] while the server is implemented in Java[6]. The communication between the modules is handled by BlazeDS[7], an extension of Tomcat designed specifically for communication between Flex and Java programs. The basic architecture of Dr. Doctor is shown in Figure 3.12.

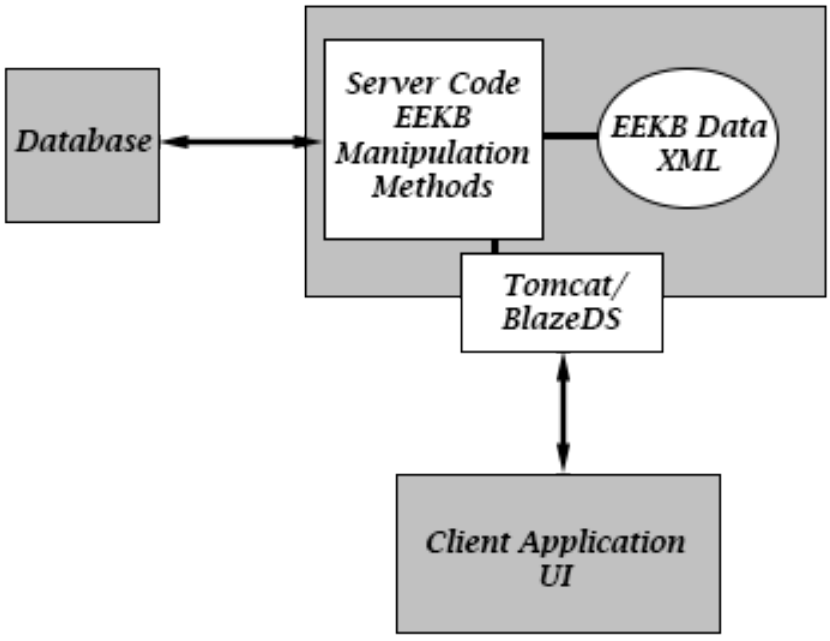


Figure 3.12: An overview of the architecture of a knowledge refinement game; a simple client-server model. However, all methods and data necessary for a KRG are stored server side, while the client application merely offers an interface with which a user can interact.

[5] *http://www.adobe.com/products/flex.html*
[6] *http://www.java.com*
[7] *http://sourceforge.net/adobe/blazeds/wiki/Home/*

It is important to note that the EEKB data, along with all methods necessary for implementing any general knowledge refinement game are located in the server module. In this way, this solution is domain-independent, and can be used out-of-the-box for anyone that wishes to quickly implement a KRG for any purpose. It becomes straightforward to implement a custom interface in lieu of the one provided here, should that be desired.

3.5 Conclusion

This chapter summarized, in detail, the technologies necessary to prove the feasibility of our approach. We described Rashi, an intelligent inquiry tutor, which is used to garner student data regarding human biology[8]. We also described some of Rashi's unique game mechanics, which we believe increases the usability of student responses.

We then described the algorithm that mines student data, and organizes it into a semantic probabilistic domain model called an evolving expert knowledge base (EEKB). We described how this model is constructed and evolves over time.

Lastly, we defined the notion of knowledge refinement games and showcase a specific example of such a game we call 'Dr. Doctor'. These games invite experts to refine the EEKB model by representing low confidence areas within the EEKB as simple questions. We described how the EEKB model can be updated and improved based on user responses to such questions.

[8] *Human biology was used in the system described here. However, all of the described technologies are domain-independent and can be applied to any ill-defined domain.*

In the following chapters, we discuss how these technologies are applied to answer fundamental hypotheses regarding the feasibility of this approach to domain model construction.

CHAPTER 4

HYPOTHESES AND METHODS

As we outlined in the previous chapter, we have built an algorithm for the automatic acquisition of expert knowledge, along with an instance of a knowledge refinement game called *Dr. Doctor*[9]. In this section, we begin by outlining the relevant hypotheses that were tested by observing behavior of the systems described in Chapter 3. We begin by outlining the hypotheses and describe in detail how each hypothesis was tested.

4.1 Hypotheses

In Chapter 3, we discussed the algorithm design for generating an Evolving Expert Knowledge Base (EEKB) by observing student data within Rashi, an intelligent interactive tutor. In Chapter 2, we described past evidence that developing an expert knowledge base for Rashi is useful. We specifically noted the ability to use such models for collaborative chat recognition and dynamic expert coaching.

Because we know that developing an EEKB is useful, we investigate the feasibility of building this model automatically. We separate our hypotheses into two general categories. The first category consists of three sub-hypotheses that generating an accurate EEKB is feasible and relatively accurate. We adapt a few terms, namely *precision* and *recall* from machine learning research[10] due to their similarity to our need and define the terms explicitly in Section 4.3 below.

[9] For the remainder of this chapter, we may refer to Dr. Doctor by its title as such, or more generally as a knowledge refinement game.

[10] Precision and recall are most notably used when discussing search results, see http://www.en.wikipedia.org/wiki/Precision_and_recall

The second subset of hypotheses expresses our belief that game-like features can be utilized to improve this general approach. We investigate two forms of game-like mechanics for this purpose. The first form describes utilizing game mechanics within the original Rashi tutor to show that students contribute more quantity and quality data. The second form describes efforts to show that knowledge refinement games are effective ways to improve the precision and recall of the generated knowledge base by engaging experts. We now list the hypotheses for our studies:

4.1.1 Generating an EEKB Automatically is Feasible

The following hypotheses are considered for testing the ability of our algorithms to acquire knowledge.

> ***Hypothesis 1:*** *Knowledge acquisition algorithms can obtain greater than 60% precision towards determination of the knowledge in an expert system within an ill-defined domain.*

> ***Hypothesis 2:*** *Knowledge acquisition algorithm can obtain greater than 50% recall on average towards the identification of knowledge in an expert system within an ill-defined domain.*

> ***Hypothesis 3:*** *A knowledge base can be built in significantly less time using our knowledge acquisition algorithms as compared with subject matter experts.*

4.1.2 The Impact of Gaming Mechanics on These Approaches

The following hypotheses are considered to study the impact of gaming tools and mechanics on improving various aspects of our automated knowledge development.

> ***Hypothesis 4:*** *Gaming mechanics in an intelligent tutor (such as the patient treatment panel) lead to a higher quantity and quality of student input.*
>
> ***Hypothesis 5:*** *A Knowledge Refinement Game used by domain experts leads to 5-10% increase in precision of the evolving expert knowledge base created by students.*
>
> ***Hypothesis 6:*** *A Knowledge Refinement Game used by domain experts leads to 5-10% increase in recall of the evolving expert knowledge base created by students.*
>
> ***Hypothesis 7:*** *Less than 5% of acquired student data is considered a mass misconception (for human biology Rashi use) as identified by a Knowledge Refinement Game.*

We posit the items above in hopes of providing evidence for an automated approach to knowledge base development. We believe that our automated approach is feasible independently and benefits from the support that targeted gaming mechanics and tools can provide.

In Section 4.2 below, we discuss the variety of data sources available for our purposes. Then, in Section 4.3, we discuss our methods in detail.

4.2 Data Collection

The data necessary for proving the aforementioned hypotheses is collected through the Rashi intelligent tutoring system (detailed in Chapter 3) and our knowledge refinement game entitled Dr. Doctor (also detailed in Chapter 3).

Over 3000 students have used Rashi since usage began in 2007. This involves many learning sessions in real classrooms across several schools, age groups, and semesters. All collected Rashi data is from live classroom settings, as opposed to controlled user studies within a lab. Also, not all participating schools provide enough computers for each student to use Rashi alone, and thus the number of registered Rashi accounts is smaller than the actual number of students who have used the system.

For the most part, Rashi users have little to no experience with case-based inquiry learning, the one exception to this rule being the students at Hampshire College. Thus, Rashi presents an opportunity for students to encounter case-based inquiry learning for the first time, and for us to evaluate our hypotheses under conditions in which student contributors are not necessarily expert in either the domain (typically human biology when using Rashi) or inquiry learning in general.

Rashi logs all student data on its server. This includes all created hypotheses (e.g., Patient has Hyperthyroidism), data (e.g., Patient has high blood pressure), and relationships (e.g., high blood pressure has no relation to Hyperthyroidism). We now discuss the various locations that have used Rashi and contributed data that we use for our study. We enumerate these schools to make clear the variety of students and schools that have used Rashi. This variety, we believe, is essential to our study as it strengthens our conclusions should our experiments succeed.

4.2.1 Hampshire College; Intro Biology

Hampshire College[11] is a small liberal arts school located in Amherst, Massachusetts; it is the flagship school for Rashi usage. Hampshire College is unique in that it employs learning strategies similar to those in Rashi. Notably, students at Hampshire engage in case-based learning throughout the entire curriculum and in almost all domains. Students are not given tests or grades. Thus, students at Hampshire College are generally familiar with the inquiry learning cycle and are, for the most part, proficient in its use.

Thus, our students from Hampshire represent the participants' most competent in Rashi's learning style and thus anecdotally provide the most clear and rich data.

4.2.2 CSI Summer Camp; Hampshire College

This summer experience, located at Hampshire College in Amherst, Massachusetts, is an opportunity for younger students to gain experience with investigative learning. Students

[11] http://www.hampshire.edu

who attend the camp are generally in middle school (12-15 years of age) and generally have very little experience with inquiry learning.

At camp, students are invited to investigate a medical diagnosis case. The most recent case (*Counselor)* asks students to investigate the disappearance of a camp counselor. The case involved investigating "real" (setup by instructors at a hiking path) crime scenes for clues, along with other tangible evidence collection activities. Students then come to the classroom and continue the investigation within Rashi, while also submitting their findings within the program.

This group of students requires significantly more guidance from an array of teachers and instructors. The instructors are careful, however, not to turn activities into series of questions or tests. Teachers merely guide student inquiry behavior so that it doesn't get too far off track.

Thus, the CSI Summer Camp students represent a unique demographic for Rashi data collection as students are both younger in age and experience the case multi-modally.

4.2.3 University of Massachusetts; Biology 101

The University of Massachusetts[12] is a large public research institution located in Amherst, Massachusetts. UMass enrolls more than 27,000 students and so many of its introductory courses are also quite large. Rashi is used as one of the labs for Biology

[12] http://www.umass.edu

101, which typically has between 300-500 students enrolled. Thus, the lab is conducted across several days and classrooms. Each classroom has its own teaching assistant for supervision, and for the most part students must share computers and Rashi accounts due to the large number of students per class.

The students at UMass are generally less versed in inquiry learning and need more guidance in order to provoke the desired behavior. Specifically, UMass students are often caught in the paradigm of right / wrong answers and thus can struggle to understand the idea that their work is strong if they engage in productive inquiry behavior. This difficulty is mitigated by our developed pedagogy that encourages teachers to provide some tight guidance and exploration of the inquiry learning to these students. This helps them feel more comfortable with the inquiry process by providing some familiar structure while still encouraging exploratory behavior.

Biology 101 at UMass is by far our greatest source of student data, accounting for more than 400 Rashi user accounts per semester.

4.2.4 Tufts University; Biology 101

Tufts University[13], a small private university located just outside of Boston, Massachusetts, and represents our first experience providing Rashi to a classroom in a completely remote unsupervised manner. Intro biology instructors contacted us about using Rashi in the classroom. We obliged, but were unable to provide any direct

[13] http://www.tufts.edu

assistance for several reasons. In particular, Tufts University was too far away for us to make trips to support Rashi use in classroom.

Thus, the data obtained from these students is from users without direct pedagogical intervention or advanced training of any kind. We were only able to provide the teachers with a short descriptive PDF summarizing some recommendations for successful Rashi use. Tufts University accounts for about 25 Rashi user accounts.

4.2.5 Summary of Student Contributions to Rashi

We now summarize student contributions to Rashi across the data collection efforts described above. In particular, we focus on contributions to four of the more highly utilized medical cases. We begin by describing the four cases in question, and then provide a full summary of the schools that utilized the given cases, along with a summary of the amount of data collected for each case.

Our studies focus on data from four Rashi cases. Each case is in the domain of human biology and is formatted as a virtually ill patient who must be diagnosed by the user. The cases are all fully unique in that each patient is defined as having exclusive issues to resolve. However, the cases are similar in that they contain medical diagnosis issues and thus students can follow a similar pedagogical procedure towards solving each. Thus, students explore the case data via Rashi's data tools (see Chapter 3), and then create and support hypotheses by utilizing Rashi's data organization tools. The cases are summarized in Table 4.1.

Reference	*Case Name*	*Description*
Janet	Janet Stone	Janet Stone has been feeling tired, distant, and is losing a lot of weight.
Angela	Angela Williams	Angela Williams appears to have an anemia problem.
Rene	Rene Beckman	Rene has been feeling apathetic for her normal sports activities.
Freshman	Frustrated Freshman	Angie just arrived at college and is feeling very ill.

Table 4.1: A summary of Rashi cases used for our efforts along with a short description of the case

These cases were used across the classrooms described in the previous section. However, the case usage is not mutually exclusive. In fact, most cases end up being used across schools, providing another layer of variability in the provided data. Table 4.2 summarizes the classrooms from the previous section, cross-referenced with the cases used at each school. You will notice that a fifth case (*Counselor*) is shown in Table 4.2, despite the fact that the data from these projects is not used for the studies in this document.

Semester	*School*	*Case(s) Used*
Fall 2007	Hampshire College; Introduction to Biology	Rene, Janet
Fall 2008		Rene, Janet
Fall 2009		Rene, Janet
Summer 2007	Hampshire College Summer Camp	Rene
Summer 2008		Angela
Summer 2009		Counselor, Janet
Summer 2010		Counselor, Angela
Spring 2009	UMass Biology 101	Janet, Freshman
Spring 2010		Janet, Freshman
Fall 2010		Janet, Angela
Spring 2011		Janet, Angela
Spring 2012		Janet
Fall 2012		Janet
Fall 2012	Tufts University; Biology 101	Janet

Table 4.2: A summary of the Rashi cases used at various institutions

The data of interest from these studies is that data which assists us in building an evolving expert knowledge base (EEKB, see Chapter 3). As described earlier in Chapter 3, students construct arguments describing their hypotheses and evidence regarding the patient's status. Thus, in order to assist us with evaluation of research hypotheses 1-3 (precision, recall and time to construct our knowledge base) we examine the following data, stored in our database in XML format:

> ***Student Hypotheses:*** *Students record in Rashi any hypothesis in consideration for a given patient. These are generally short pithy statements regarding the patient's health (e.g., Patient has Hyperthyroidism).*
>
> ***Student Data:*** *These items consist of observable data in Rashi. Some of these items are collected directly from Rashi tools (e.g., Patient's recorded blood pressure) and others are purely student created (e.g., Patient seems nervous).*
>
> ***Student Relationships:*** *Relationships are semantic connections between any two hypotheses and/or data. Generally these links related observable data to their consistency with one or more hypotheses (e.g., Blood iron levels SUPPORTS Patient has hyperthyroidism).*

Thus, the data above, across hundreds of Rashi accounts provides the necessary data to test our knowledge acquisition algorithm over four distinct Rashi cases. In particular, we were able to collect thousands of data points for each case. We find that on average, each individual student creates about 15 unique data points. Table 4.3 summarizes each case, and the total number of data points created in Rashi for each case. Because of the storing

techniques for this data, it was very difficult to separate the data and relationship counts, and thus they are shown in Table 4.3 in sum.

Case	***Num Hypotheses***	***Num Data / Relationships***
Janet	4144	8108
Angela	1133	2424
Rene	3400	5978
Freshman	329	706

Table 4.3: Summary of data point totals for each case. This data is used to automatically construct an evolving expert knowledge base. Data and Relationships were stored in the same database table, and thus are shown in total.

Thus, we have an extensive corpus of student data with which to work, over four unique cases. This provides the perfect platform upon which to study the effects of our knowledge acquisition algorithm. We will discuss in Section 4.3 exactly how this data is used.

The data used for our research hypothesis 4 (game-mechanics lead to higher quality/quantity of student input) is a subset of the data above. In particular, we select two student data sets that are as similar as possible with one distinct difference. This difference is the presence of a new Rashi game feature, particularly the patient status panel and the treatment feature (see Chapter 3 for more details on these features). For this purpose, we select two of our spring semester UMass Biology classes. The spring 2011 class did not have these new Rashi game features, while the spring 2012 class did.

4.2.6 Knowledge Refinement Game Data Collection

Research hypotheses 5-7 address the effectiveness and efficiency of the knowledge refinement game (KRG). In order to investigate these effects we enlisted the help of seven graduate student volunteers at the University of Massachusetts. These graduate students were "near" experts in biology and all have experience teaching Rashi labs to undergraduate students. They worked within the KRG Dr. Doctor with prior knowledge of Rashi and its basic usage, and therefore had knowledge of the context from which the questions in the KRG originated.

These experts were asked to answer approximately 100 questions (plus or minus a few questions was permitted). The participants were given a $10 gift certificate for answering 100 questions, and were offered a "Special Prize" if they achieved the highest score. Recall, from Chapter 3, that Dr. Doctor offers points for answering questions, including bonus points for high numbers of contributions or agreeing with other user's answers.

Table 4.4 below summarizes the expert contributions and their individual contributions within the knowledge refinement game. In particular, we see that seven experts in total and several experts (three out of seven) answered more than 200 questions instead of the required 100. Additionally, two out of the seven experts did not reach 100 questions. We do know, however, that expert seven was not able to continue past 71 questions because the knowledge refinement game ran out of questions at this point, forcing the expert to cease. Thus, it appears that only one expert (expert 6) did not successfully reach 100 questions.

User Alias	*Number of Contributions*
Expert 1	365
Expert 2	104
Expert 3	456
Expert 4	165
Expert 5	250
Expert 6	56
Expert 7	71
Total	**1467**

Table 4.4: Users and contributions per user to Dr. Doctor data collection

The data above show that a total of 1467 questions in the KRG were answered, and thus provides a sufficient data set for studying the efficacy of the KRG. Additionally, the game exports an XML data file containing the adjusted EEKB after every 15 answered questions. Thus, these data files provide us with the opportunity to study the knowledge refinement game's effects over time.

In Chapter 5, we describe the negative results provided by these seven contributors. This led us to carefully consider subtle adjustments in the design of the knowledge refinement game that could alleviate these poor results. In short, our investigation led us to remove specific features of Dr. Doctor and to execute the data collection a second time. During this second iteration, we were able to gather three of the original seven participants to ask them to contribute an additional 100 question responses. The participants contributed 250, 235, and 105 additional contributions respectively.

Lastly, the knowledge refinement game's database stores a record of every question that is asked along with pertinent data. The information stored is as follows:

Id: *Unique identifier for this particular question.*

Question Type: *The category of question that was asked (e.g., Verify a node, or verify a relationship).*

Item Data: *Data necessary to track the node or relationship in question along with the system's confidence. By confidence we mean the confidence within the EEKB that this item represents true knowledge (see Section 3.3.1).*

User Response: *The answer provided by the player (usually Yes or No).*

This data is used to verify the EEKB's accuracy and note the presence of any misconceptions (i.e., when confidence of the system is high but the players argue that the node or relationship is in fact false).

4.3 Methods for Evaluating Hypotheses

We now present our methods for utilizing the data described above in an effort to provide support for our hypotheses. We begin by defining some terms and then move on to describe the process for analyzing data.

4.3.1 Definition of Precision and Recall

We evaluated Hypotheses 1-2 by measuring the precision and recall of our automatically generated knowledge base (EEKB). To judge the quality of our EEKB, we compare it directly to our expert created knowledge base created by subject matter experts (which is

preserved for the purpose of performing this comparison). We refer to this human generated expert knowledge base as the HEKB. We utilize two distinct but related metrics. The first, called 'precision' judges the truth of the generated knowledge with little regard to its breadth. The second, called 'recall, is the same as accuracy but takes into account the breadth of knowledge that is created.

We define precision as the information in the automatically generated graph that is true. Thus, the precision of a generated graph will be 100% if the generated graph is any subset of the expert generated graph. The formula for assessing precision of a generated graph EEKB compared to a human created graph (HEKB) is:

$$Precision\ (EEKB, HEKB) = |\ EEKB \cap HEKB\ |\ /\ |\ EEKB\ |$$

Where EEKB is the automatically generated graph and HEKB is the human generated graph

The cardinality of a graph includes the set of nodes and relationships. This calculation is, of course, predicated on the assumption that an expert created graph is always filled with accurate information.

Recall is the percentage of the human expert's knowledge base (HEKB) that we have successfully generated. Thus, this is a measurement of how well we have included the breadth of knowledge created by a human expert.

$$Recall\ (EEKB, HEKB) = |\ EEKB \cap HEKB\ |\ /\ |\ HEKB\ |$$

Where EEKB is the automatically generated graph and HEKB is the human generated graph

4.3.2 Calculating Precision and Recall of an EEKB

In order to calculate the precision and recall of a given EEKB, provided that we have an HEKB with which to compare, depends on our ability to calculate the intersection of these models. This can be done manually, but doing so would require an extensive amount human grading time that, given our available resources, is unreasonable. Thus we instead utilize an automated approach to calculating this intersection. The pseudo-code for this process is presented in Figure 4.1 below:

```
Define intersectionSize:
    For each node 'n' and relationship in EEKB
        Search HEKB for exact match to n (String equality)
        If a match is found:
            Increment intersectSize counter
        Else use text search engine to search for pseudo match
        If search engine returns match with score > threshold
            Increment intersectSize counter
    End For
    Return intersectSize counter
End intersectionSize
```

Figure 4.1: Pseudo-code for calculation the intersection size of an EEKB and a related HEKB

Of particular note in Figure 4.1 is the use of a text search engine and a match threshold with which to filter results from said engine. If an exact match is not found (via String equality), then our module needs to be able to identify semantic matches that are not syntactically equivalent (i.e., 'Hyperthyroidism' and "Patient has hyperthyroidism' are the same hypothesis, but are not written the same way). For this purpose we utilize the Lucene search engine[14].

[14] http://lucene.apache.org/core/

Lucene returns items along with a match score. This score is higher if the match is stronger. The issue of course is the occurrence of false positives, in which two nodes that are not semantically equivalent are detected as being so. To mitigate these effects, we determined a minimum match score needed by Lucene to ensure a high level of match accuracy. We hand graded a series of 188 Lucene matches during an execution of our node intersection function to verify that our threshold was wisely chosen. The results, given our eventually chosen threshold value of 3.5 (a number which seems arbitrary to those who have never used Lucene), are given below in Table 4.5:

Total	***Correct***	***Incorrect***	***Percent***
188	185	3	**98.4**

Table 4.5: Results that demonstrate the correlation between hand grading student answers and automatic high search results threshold. The high results for correct answers verify the efficacy of an automated calculation of precision and accuracy.

Thus we are able to verify that results of our calculations (see Chapter 5 for results) are generally strong estimations of the actual values. This is because our selected match confidence threshold, when searching with Lucene, is high enough to ensure a 98 percent success rate as compared with human grading. Although there may very well still be a small handful of false positives, these should be extremely minimal and mitigated further by the presence of small sets of false negatives.

Thus, we are able to automate the calculation of the intersection of an EEKB and an HEKB, which in turn allows us to quickly calculate accurate values of the precision and recall for the EEKB.

4.3.3 Precision and Recall Over Time

Instead of simply observing the precision and recall of our generated EEKB after processing all student data, we observe how the generated model changes over time. We start by exported all student data to a large xml file. This file describes student creation of hypotheses, data, and relationships in true chronological order. Thus, this xml file represents a log of student interactions in Rashi (or rather the particular interactions we are interested in) that can be analyzed and re-simulated.

We then wrote a Java program that parses through this xml and simulates students working in Rashi. These student actions are fed into our EEKB generation algorithm and the generation module (see Chapter 3) constructs the model. We are then able to take snapshots of the state of the EEKB at any time and calculate its precision and recall. We decided to take such a measurement after every 100 student inputs are considered. We also record some simple but potential useful statistics regarding the EEKB. These include its size, number of nodes, number of relationships, and other similar values. At the end of execution, these values are automatically exported to a spreadsheet for data analysis.

This provides the necessary information for analyzing the efficacy of our knowledge acquisition algorithm.

4.3.4 Calculating the Efficiency of EEKB Construction

In order to evaluate hypothesis 3 (an EEKB can be built more rapidly than one implemented by human subject matter experts) we compare efficiencies of various knowledge base construction methods. We first calculate the total number of hours necessary to build such a model. We begin by estimating the HEKB generation time, which can be done easily by asking our subject matter expert and others who worked on the project during that time. Our SME estimated that it takes 600 hours for a team of two partners to create 1000 components. Thus, this represents a total of 1200 human hours per 1000 components.

Estimating the efficiency of EEKB construction is slightly trickier, especially when considering how heavily parallelizable is the process. We began by examining the size and recall of the generated EEKB over time along with the amount of data necessary to produce it. We posit that EEKB recall over time is logarithmic in nature and thus levels off. We examined this curve and determined a reasonable estimate for the amount of student data necessary to obtain a saturated amount of EEKB data.

Once we know the amount of student data necessary to saturate an EEKB, we simply divide that by our observation of the number of hours spent by the students (generally about 1.5-2 hours per student per case). This data was obtained from our database and is summarized in Table 4.6 below:

Data Type	***Data per Hour***
Hypothesis	3
Data	8
Relationship	4
Total:	**15**

Table 4.6: Average student data creation per hour

Utilizing the values in Table 4.6, we were able to calculate the number of student hours within Rashi necessary to create a saturated EEKB. More details on this analysis (and specifically the results from it) can be found in Chapter 5.

4.3.5 Analyzing Effects of Rashi Game Features

Our first goal for proving hypothesis 4 (gaming mechanics lead to a higher quantity and quality of student data) is to provide evidence that game mechanics in Rashi lead students to produce more data in Rashi. To study this possibility, we selected two data sets that shared a maximum number of qualities while differing only in the presence of game mechanics. In particular, the mechanics in question are the patient status panel (providing a health/urgency game mechanic) and the treatment functions (which provides diagnosis and a sense of player reward).

The data sets we have that fit these qualities are the Spring 2011 and Spring 2012 UMass Biology 101 classes. These classes were similar on almost every other respect. They were taught by the same teachers, used similar pedagogies, and were used by a similar caliber of student. The only noticeable difference between the two groups is the presence

of game mechanics within Rashi. The 2011 class did not have game mechanics while the 2012 class did.

Our methodology here was straightforward. We aimed to show that the 2012 class produced significantly more Rashi data than did the 2011 class. We study this by simply observing and comparing the produced data from each class. In addition, we aim to show that the quality of work is higher when game mechanics are present. We accomplished this by estimating a grade for each student in the respective experimental groups. This grade must be estimated because of the ill-defined nature of Rashi.

Our estimated grade for students incorporated four features of student arguments that are representative of strong inquiry behavior. These characteristics include breadth of explored hypotheses, depth of exploration into each hypothesis, quality of chosen hypotheses, and quality of established relationships. We chose to exclude data variables because virtually all data collected in Rashi comes from the systems interaction tools, which simply leads to an inflated score for all users. The estimated grades are calculated as follows:

$$\text{Grade} = [\text{Correct}(H) / |H|]*W1 + [\text{Correct}(R) / |R|]*W2 + [|R| / |H|]*W3 + [|H|]*W4$$

Where:
H = the set of student hypotheses

R = the set of student relations

Correct: a function that returns the number of items in the input set that match to the expert knowledge base

W = A weight (0<=W<=1) for each term of the grade. W1+W2+W3+W4 = 1.0

Using this formula we are able to estimate the quality of work for any given student and after generating these grades, we can compare the two experimental groups to determine if there is a difference in work quality within Rashi when game mechanics are added to the system.

4.3.6 Analyzing Effects of Knowledge Refinement Game

In order to provide support hypotheses 5 and 6 (increased precision and recall in the EEKB provided through use of the KRG) we measure the impact on then EEKB. Dr. Doctor produces XML descriptions of the EEKB's state after every 30 expert contributions as mentioned earlier in Section 4.2.7, This provides the data necessary to observe the game's effect on our EEKB model over time. We wrote a Java script that imports these XML files in sequence. The script then executes the same methods used for calculating precision and accuracy for each successive EEKB instance.

The script outputs a spreadsheet with this data for each EEKB instance, allowing us to analyze how usage of Dr. Doctor impacts the quality of the knowledge base. This information is sufficient to draw conclusions regarding hypotheses 5 and 6.

4.3.7 Verifying EEKB Quality / Detecting Misconceptions

In order to provide support for hypothesis 7 (the RKG enables detection of student misconceptions) we compare knowledge statements inputted by students that are identified by experts as true. We use the Dr. Doctor database stores records of all

questions asked along with necessary data and the response given by the experts. This provides us with two unique opportunities.

The first is an opportunity to verify, via an orthogonal metric, the quality of the EEKB generated by students. We do this by simply observing the percentage of EEKB items in which our experts agree that the knowledge is accurate. We can then compare these results to our precisions and recall calculations to verify quality.

Additionally, this data provides an opportunity to study student's misconceptions. It is not unexpected that students may contribute data that is false but generally considered true (i.e., a misconception). We examine the presence of misconceptions by searching the logs for instances in which an EEKB entry had high confidence (i.e., many students agreed it was true) but was rated as false by expert Dr. Doctor players. We posit (via hypothesis 7) that the numeration of these instances is low (less than 5 percent).

4.4 Conclusion

In conclusion, we have outlined in this section our hypotheses regarding the efficacy of our approach. In particular, we posit that automatic acquisition of domain knowledge is feasible when observing student behavior within Rashi, leading to promising values for both precision and recall. We discussed in detail how we calculated these metrics algorithmically and how our knowledge refinement game provides the data necessary to verify such results.

We also outlined our data collection efforts relevant to the hypotheses in question. In particular, we have garnered data from a vast array of classrooms over several years. These classrooms represent variations in age, level of teacher engagement, level of researcher engagement, and many other factors. Thus, the data incorporated into our generated EEKB originates from a well-randomized pool of contributors.

We then discussed our evaluation procedures. In sum, we simulated several years' worth of student input on Rashi, feeding data into our knowledge acquisition algorithm and exporting relevant statistics as they evolved over time. We then used the final generated EEKB for our experiment with a knowledge-refinement game called Dr. Doctor. We discussed how Dr. Doctor stores relevant data and how this information is used to study our remaining hypotheses.

In the next chapter we discuss the results of the evaluations described in this chapter.

CHAPTER 5

RESULTS

In this chapter, we summarize the results of the analysis described in Chapter 4 in an attempt to provide evidence for or against our hypotheses. We begin by showing the results of simulating student input through our evolving expert knowledge base (EEKB) generation algorithm. We see that the algorithm produces EEKB models with acceptably high precision. We also observe that the recall of these same EEKB models is relatively low, and provide brief insight as to why this is the case.

We then show our results of EEKB construction efficiency. We find that our algorithm is both more efficient than construction with human experts and is more heavily parallelizable. We then discuss the effects of Rashi game mechanics on student contributions, which we discover lead to significantly more student input as well as higher quality input. We conclude by summarizing our findings of using the knowledge refinement game (KRG) to improve our EEKB. An initial study of the KRG provided limited results, after which system changes were made that resulted in greatly improved results.

5.1 EEKB Generation Results

As discussed in the previous chapter, over 15,000 student data inputs across four unique Rashi cases were fed through our knowledge acquisition algorithm. This section discusses the EEKB models that were produced as a result of this process. We discuss

their sizes and then their quality. We also discuss how our measures of EEKB quality change over time as additional student input was considered.

5.1.1 Raw EEKB Data

After executing our script described in Chapter 4 on the corpus of student data, we obtained four evolving expert knowledge bases (one for each Rashi case). The exported EEKB only includes items (nodes or relations) whose confidence values exceed a set threshold (set at 30 percent confidence). For example, the generated model is stripped of many instances of student input that occurred only once. The sizes of these models is shown in Table 5.1 below:

Case	***~Student Data Considered***	***Nodes***	***Relations***	***Total (Nodes + Relations)***
Janet	12000	234	76	**310**
Angela	3500	158	8	**166**
Rene	9300	266	35	**301**
Freshman	1000	54	15	**69**

Table 5.1: Sizes of generated EEKB models relative to amount of student input that was considered.

We see that generally, more student input leads to a larger EEKB, which is intuitive and not particularly surprising. We also notice that our set of relations is relatively low compared to our node sizes. Students contributed many potential relationships in our original data set, but there was higher disparity in the relationships provided and thus less student agreement. A relationship needed to reach a confidence of 30 percent to be included in the final EEKB, and thus the disparity described leads to relatively small sets of relationships in our final EEKB.

We then measure precision and recall of each of our four generated EEKB models. These results can be seen in Figure 5.1 below. We also analyze 'Node Recall' and 'Node Precision'. These values are precisely the same as precision and recall (see Chapter 4 for calculation formula), but differ in that they only consider the nodes (and not the relations) within the EEKB. Thus, node recall is the number of nodes in both the EEKB and the HEKB divided by the number of nodes in the HEKB. Likewise, node precision is the number of nodes in both the EEKB and HEKB divided by the number of nodes in the EEKB.

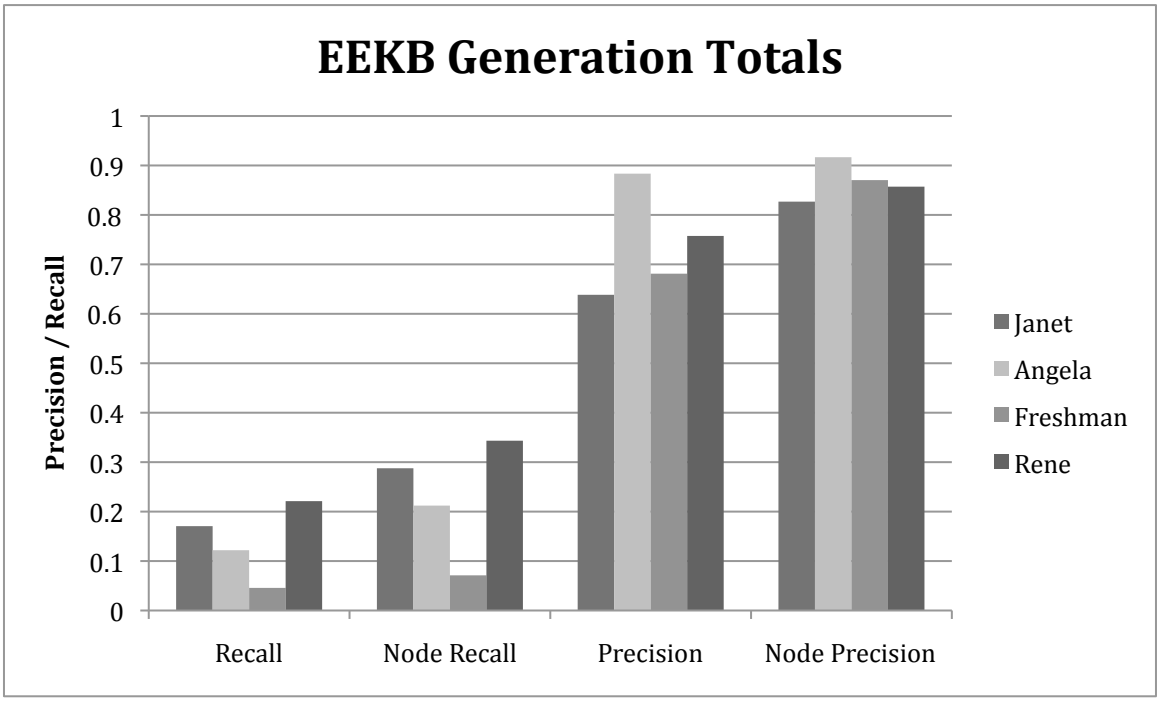


Figure 5.1: Precision and recall of the final EEKB generated models. We see that precision is consistently above 60 percent, while recall is generally between 5-25 percent.

In general, precision is consistently over 60 percent, which we consider high. This means that of the knowledge generated in our EEKB, more than 60 percent of it is accurate, 'true' knowledge. Additionally, our best results yield just below 90 percent precision.

We include the values for 'Node Precision' as evidence that our nodes alone actually obtain even higher values, yielding greater than 80 percent precision for all four cases.

Our recall values, on the other hand, appear to be very low. For our case with the least amount of data (Freshman), the EEKB only yielded about 5 percent recall. In the best case, we yielded just above 20 percent recall. Although this result may seem like a negative one, prior results actually reveal a different story. We have seen in previous studies that students typically only explore somewhere between 10 and 20 percent of the human crafted knowledge base (HEKB) [109]. Thus, recall values between 10 and 20 percent should not be considered unreasonable because, in fact, the students in our studies are exploring the same knowledge as were studied by students in a previous study (HEKB) [109]. This previous study discovered that students only explore about 10-20 percent of the knowledge base, and that algorithms that use the knowledge base to reason and coach students can do so more effectively by only utilizing a smaller portion of the model.

5.1.2 Precision and Recall Over Time

Our next step was to sample these EEKB models during the generation process. We obtained EEKB samples after every 100 data considerations, calculating the precision and recall of each graph. The results of this analysis (for precision only) are shown below in Figure 5.2. Note that the multiple lines in the chart represent the four Rashi cases for which we used student data. However, some of these lines appear shorter in correlation with the amount of student data that was considered. We see that the Janet Stone case

has the longest plotted line due to the more than 11,000 data points available for that particular case.

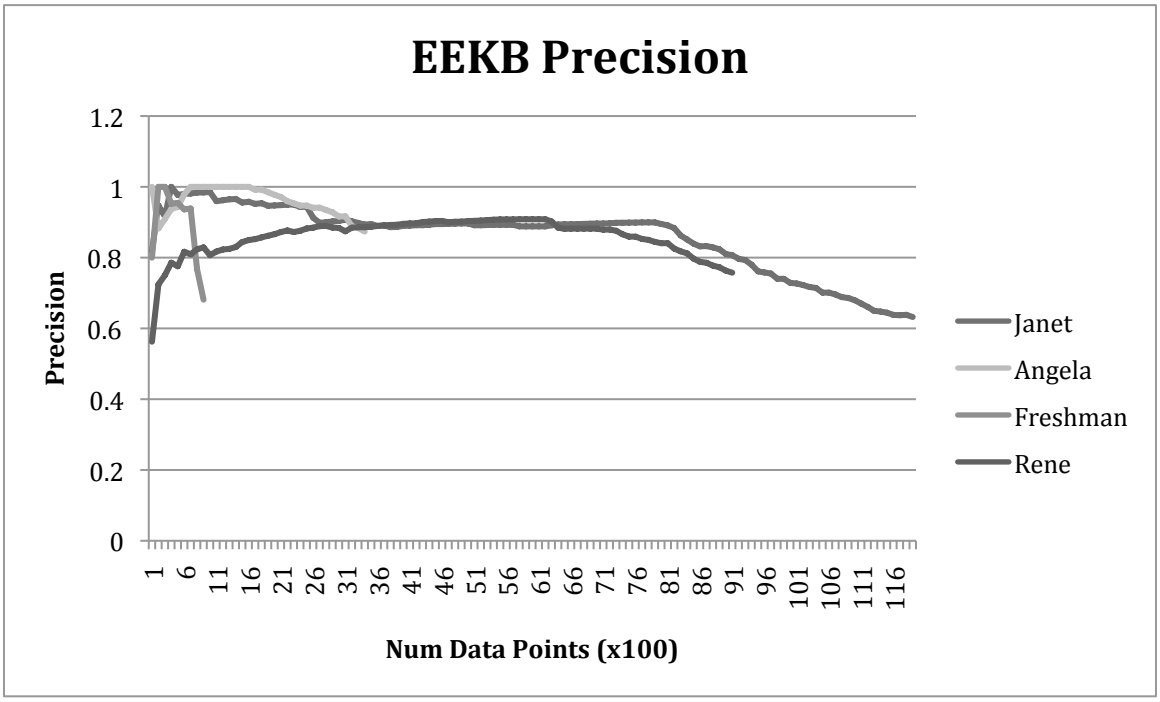


Figure 5.2: Precision of EEKB models over time. Each plotted point represents a precision sampled after an additional 100 pieces of student data were fed through the knowledge acquisition algorithm.

We see that precision remains above 60 percent for the entire lifetime of all EEKB models that we considered. We also see a high level of variability in precision during the first 1000 data points or so. This variability is sensible because of the nature of the precision calculation. When an EEKB is small, each piece of the model accounts for a higher portion of the given precision score. Thus, we would expect to see high variability in precision for small knowledge bases because each item carries significantly more weight.

We also observe that the precision of our EEKB models tends to level off starting at around 2000 student inputs. This observation only holds for three of the four knowledge bases because one Rashi case (Frustrated Freshman) received less than 2000 input data points from students. On a positive note however, this leveling occurs at around 90 percent precision, which is significantly higher than our expectations.

For most cases, we also see a precision drop off towards the end of the data set. This occurs because students tend to collect all their hypotheses and data before establishing relationships. Thus temporally, precision drops off when student inputs start to heavily favor relationships instead of additionally hypotheses or data. There are several reasons we feel that this drop off occurs. The first is the exponential growth that occurs in the number of potential relationships when adding additional nodes. Our expert's hand crafted knowledge base (HEKB) does not consider every possible relationship because of this scale, and thus it becomes more likely that student inputs are legitimate while not being represented in the HEKB. There are other potential causes for this precision decrease, which are explored in length in Chapter 6.

We also calculated EEKB recall over time in the same manner as described for precision. The results of this analysis are plotted in Figure 5.3. As was described in Figure 5.2, the cases with greater student data consideration contain longer plotted data.

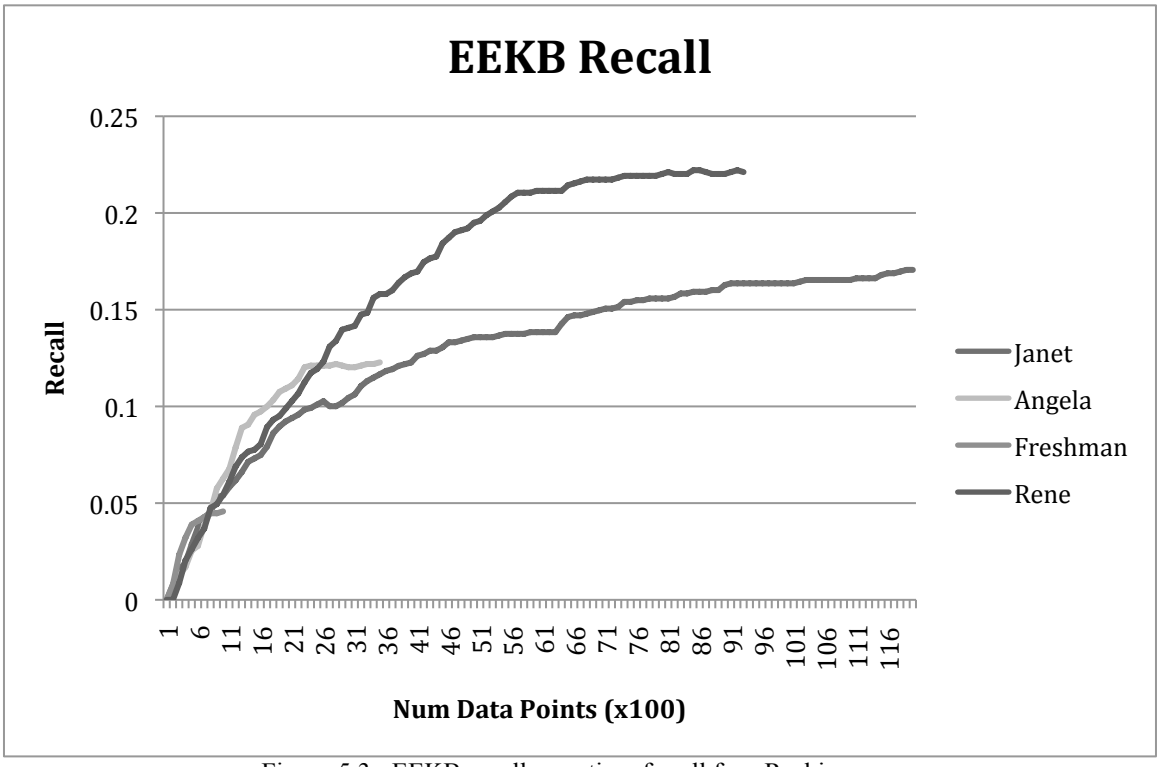


Figure 5.3: EEKB recall over time for all four Rashi cases

We see that EEKB recall is monotonically increasing over time. This is expected behavior when considering the calculation for recall, which is not penalized when items in the EEKB are superfluous or inaccurate. Thus, as EEKB models grow over time, recall can only increase or remain steady. We also observe that all four cases follow approximately the same trajectory, and are logarithmic in nature. This is sensible because of the nature of student input. There is a relatively finite amount of information that large groups of students discover and insert into Rashi. Thus, as we collect and incorporate this information into an EEKB over time, the probability of incorporating a unique piece of data from a new student steadily declines.

We also observe here that the two cases with the lowest final recall values (Freshman and Angela) are also the cases that received the least amount of student data. We see that the

recall graphs for these cases follows a similar trajectory to the others while unfortunately stopping short of the more data heavy counterparts.

5.2 Efficiency of EEKB Generation

In this section, we present our results of calculating the efficiency of generating an EEKB automatically. Our methods for doing so are aided by the logarithmic nature of EEKB recall that we presented in the previous section. We begin by defining the notion of EEKB saturation. We consider an EEKB to be saturated when 1000 new student data inputs only yields one percent additional recall to the EEKB. Thus, the recall graph is flat for 1000 student entries.

To discover the points at which our EEKB models are, or in some case were likely to be, saturated, we fit a logarithmic curve to each recall graph. The graphs of the recall of each individual Rashi case, along with their logarithmic fits, are presented in Figure 5.4. To calculate the saturation point for each case, we simply take the derivative of the functions defining the best-fit curves for each graph. We deduce that when the derivative of these functions is equal to 0.001, then the EEKB must be saturated because of the small nature of the fit's slope.

Thus, we determined the number of data points necessary to saturate an EEKB for each Rashi case considered. Even for those cases with limited available data, we were able to simply extrapolate the fitted curve. We then obtain four values (one for each case) that determine the saturation point for that case. We averaged these values together to obtain

an average estimate of data necessary to saturate a generated EEKB. The saturation values obtained are summarized in Table 5.2. We see that this average saturation point occurs at approximately 4,510 student contributions.

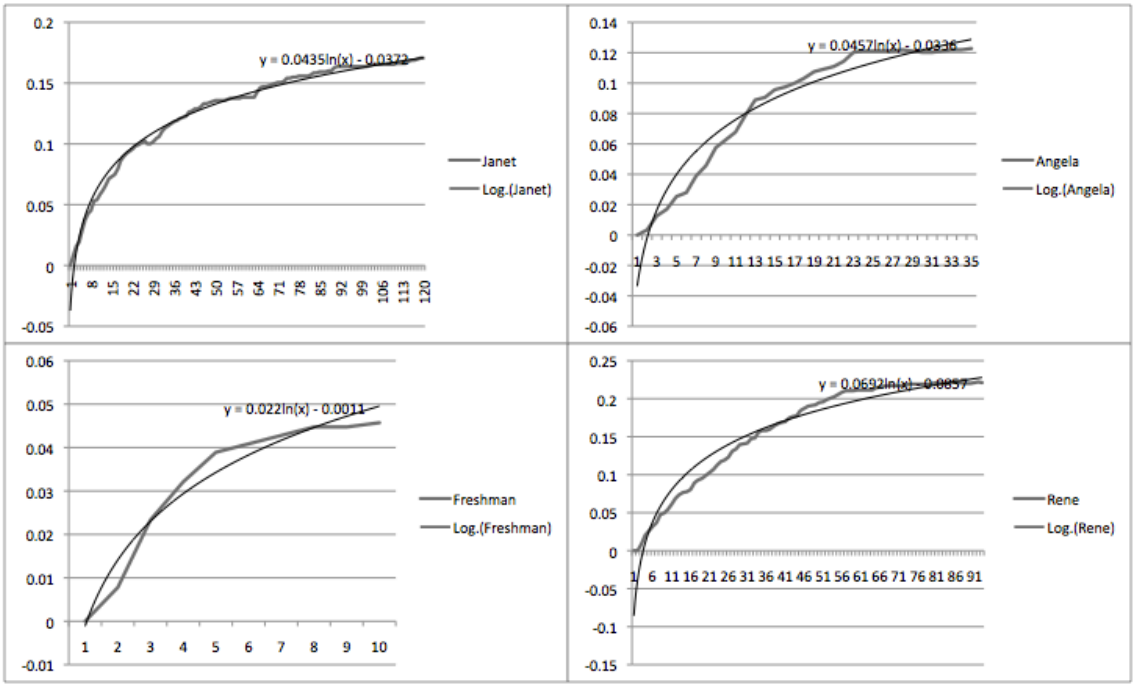


Figure 5.4: Recall over time for four Rashi cases individually, along with best logarithmic fit curves.

Additionally, in order to calculate the efficiency of EEKB generation, we need an accurate estimate of how much student data is created per hour of student work. We analyzed our Rashi data to determine the average number of hypotheses, data, and relationships that are created by Rashi students for any given hours worth of work. Details on these results are available in Chapter 4, but are included again for reference in Table 5.2 below.

EEKB Case	Num Data Points Until Saturation
Janet	4350
Angela	4570
Rene	6920
Freshman	2200
AVG:	**4510**

Avg Student Hypos / Hour	3
Avg Student Data / Hour	8
Avg Relations / Hour	4
Avg. Stud. Data / Hour	**15**

Table 5.2: Number of student inputs necessary to saturate an EEKB for each given Rashi case (left) along with a summary of the amount of data students contribute within Rashi on average per hour (right).

Now that we have garnered the information in Table 5.2 above, the calculation of EEKB efficiency becomes trivial. By simply dividing the data necessary for a saturated EEKB by the amount of work produced by students per hour, we obtain a strong estimate for the number of hours necessary to build a knowledge base, as shown in Table 5.3.

EEKB Saturation	Stud. Data / Hour	Num. Hours
4510	15	**300.67**

Table 5.3: Calculation of number of hours of student work necessary to saturate an EEKB

We see that 300 hours of student work is necessary to produce a fully saturated EEKB, while a reasonably saturated one surely requires less time. It is important to note though that there are several variables in play here, namely the number of students, how long their Rashi sessions last, and how many unique classes are available. Table 5.4 below summarizes a simple application of this concept by describing how fewer students are necessary, in theory, when students use Rashi for longer periods of time.

Session Length (Hours)	Students Needed
1	300.67
2	150.33
3	100.22

Table 5.4: Calculations of number of students necessary to generate enough data to saturate an EEKB, shown over multiple session lengths with Rashi.

Thus, we conclude that a class of 300 students can generate a fully saturated EEKB over the course of a one-hour session. In other words, we can populate an EEKB in around one hour with 300 students. Of course, it is not necessary that these students use Rashi simultaneously. Any combination of student class size and number of sessions can be utilized, so long as the total number of student work-hours totals 300. This begins to illuminate the potential for a heavily parallelizable solution to generating expert knowledge bases, because a class of 300 students can generate a precise EEKB in merely one hour. However, in order to properly compare human generated approaches with our automated one, we must normalize for both the number of knowledge components generated, and the degree of parallelization utilized. For the HEKB, our subject matter estimated a required time of 60 hours for each of two team members in order to build 100 components within a single Rashi case. Our HEKB models store 1000 components on average, which yields approximately 600 hours of work (as a team) or 1200 total human hours of work for 1000 components. Thus, because our HEKB required 1200 total human hours to produce 1000 components, this represents a build rate of 1.2 hours per component. The EEKB build rate, given 300.67 hours to produce 310 components reveals a build rate of 0.97 hours per component. This EEKB construction time reflects a 20 percent increase in efficiency. Student work is heavily parallelizable, and thus a class of 300 participants working during a single session would create a saturated EEKB in merely one hour of Rashi usage. Table 5.5 provides a more accurate comparison of the approach to building an HEKB versus an EEKB and the efficiencies of building the models both in series and in parallel. We assume a classroom size of 30 students using Rashi in parallel on average.

	Components Built	*Hours*	*Num Contributors*	*Hours per Component (non parallel)*	*Hours per Component (parallel)*
HEKB	1010	1200	2	1.188	0.594
EEKB	310	300.67	30	**0.970**	**0.032**
			% Decrease:	***18.37***	***94.56***

Table 5.5: A summary of build times necessary for an HEKB and respective EEKB. EEKB models can be built more efficiently using a combination of available students and our knowledge acquisition algorithm.

We observe that our automated approach generates improvements in efficiency when knowledge generation is considered both in series and in parallel. When in series, our algorithm produces knowledge components almost 20 percent faster. However, the automated approach performs best when leveraging the vast quantities of students that are generally available (as opposed to the number of available experts). When assuming an average classroom size of 30 students in parallel, our algorithm generates knowledge 95 percent faster than can expert humans. Additionally, it is important to note students were not asked to perform any tasks outside of their default schedule (i.e. they were going to use Rashi as a learning tool anyway in their classroom). Human experts, on the other hand, are required to contribute a significant amount of additional time outside of their default responsibilities.

5.3 Effects of Rashi Game Mechanics

The positive results presented above are partially dependent on students contributing an effective amount of student data during their work sessions in Rashi. Thus, it is clearly in our interest to foster a learning environment that is conducive to maximizing student contributions. In this section, we examine our hypothesis that game mechanics within Rashi can lead to more productive student behavior. In Chapter 3, we described some

simple game mechanics that were added to Rashi. In particular, we added a patient status panel that associated a health bar with each patient. We also implemented a patient treatment feature that allowed students to apply treatments and observe how the patient is affected. For more details on these features, see Chapter 3.

We were able to select two datasets that were as similar as possible except for these game mechanics. The datasets in questions both come from the UMass Biology 101 classrooms in 2011 (no game features) and 2012 (game features). An analysis of the work produced by students in each of these datasets is presented in Table 5.6 below:

Year	*Num Projects*	*Num Data / Relations*	*Num Hypotheses*	*Total*	*Contributions / Student*
2011	396	4342	2111	6453	**16.295**
2012	539	9328	4683	14011	**25.994**

Table 5.6: Amounts of work contributed by students in the same college course in successive years. 'Num Projects' refers to the number of Rashi user accounts. We use this terminology because students, in some cases, work in groups.

We see that the group that was given the game mechanics contributed significantly more work per student than the control group. Namely, we see a 59.5 percent increase in the raw amount of data contributed by students. We see that the control contributes the same approximate amount of work per student that we had observed in our previous efficiency analysis (~15 contributions per student). In contrast, students who were given Rashi with game mechanics contributed about 26 unique knowledge entries each. A summary of our efficiency analysis, assuming 26 contributions per hour is shown below in Table 5.7:

EEKB Saturation	***Stud. Data / Hour***	***Num. Hours***
4510	26	**173.46**

Table 5.7: EEKB saturation efficiency given the increased contributions observed by users who were given Rashi with game mechanics

Thus, it appears that adding game mechanics to Rashi has the potential to decrease the number of student hours necessary to saturate an EEKB by up to 59 percent. This represents a vast improvement in efficiency. Table 5.8 repeats the analysis from Table 5.5 substituting an increased efficiency for EEKB generated that is observed when student are provided with game mechanics in Rashi.

	Components Built	*Hours*	*Num Contributors*	*Hours per Component (non parallel)*	*Hours per Component (parallel)*
HEKB	1010	1200	2	1.188	0.594
EEKB	310	173.47	30	**0.560**	**0.019**
			% Decrease:	***52.86***	***96.8***

Table 5.8: A summary of build times necessary for an HEKB and respective EEKB. EEKB models can be built more efficiently using a combination of available students, our knowledge acquisition algorithm, and game mechanics in Rashi.

We observe that in serial, knowledge generation efficiency can be improved by 53 percent. In addition, the efficiency improvement when considering work done in parallel rises to 97 percent.

It is in our interest also to show that this increase in work is not met with a decrease in the quality of that work. To analyze this factor, we calculated a grade for all students in each experimental group. Because inquiry learning is naturally subjective, this grade is an estimation of the quality of work done within Rashi by a particular student. For more

details on how this grade is calculated, see Chapter 4. Table 5.9 summarizes the key statistics related to the grades of our two groups in question.

	Control	Experimental
Min	0.00	0.00
Min Valid	0.00	0.00
Q1	0.29	0.42
Average	0.44	0.59
Median	0.43	0.71
Q3	0.72	0.85
MaxValid	0.93	0.93
Max	0.93	0.93
	P:	***1.12E-11***

Table 5.9: A summary of the key statistics related to the grades of students who used Rashi without gaming mechanics (Control) and with gaming mechanics (Experimental).

We see that the group that was provided with game mechanics significantly outperforms the group that did not receive game mechanics.

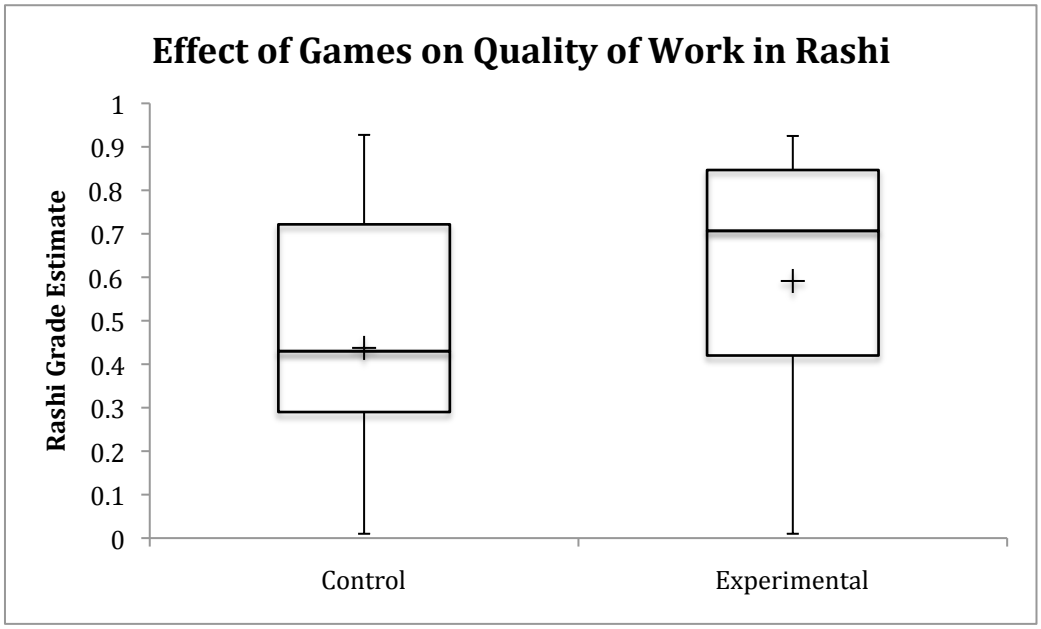


Figure 5.5: Visual representation of the improvement in "grades" for students when game mechanics are introduced into the Rashi system (Experimental Group).

This means that game mechanics may not only increase motivation and engagement, but can also be applied as the mechanism for increasing promising inquiry learning behavior.

This information is presented graphically in Figure 5.5. We see that the minimum and maximum values for each dataset are unchanged, while the quartiles, median, and average are all improved when game mechanics are introduced with Rashi.

5.4 Verifying EEKB Accuracy Using KRG Data

We now present the results of using the knowledge refinement game Dr. Doctor to refine and improve the quality of the EEKB. As summarized in Chapter 4, we invited our experts (graduate teaching assistants) to refine the expert system developed by students for the Angela Williams case. Seven experts contributed 1467 total answers by playing Dr. Doctor over the course of one week. We describe an analysis in which we use the data obtained from this experiment to validate the quality of the original EEKB developed by students.

We do this by analyzing the answers provided by experts to the questions that involved verifying nodes or relationships within the EEKB (See Chapter 4 for details on the KRG question types). If one expert agrees that an EEKB entry is true knowledge, then we consider that entry confirmed. Otherwise we consider the entry denied (and it is subsequently removed if other players agree). We present our findings of this analysis in Table 5.10 by summarizing the degree to which EEKB generated items were deemed accurate. We also include the average confidence values of the respective entries in an attempt to observe differences in the qualities of accepted versus denied entries. Confidence values here are transferred directly from the originally generated EEKB and are based on the number of students who each contributed a single entry to the

knowledge base. For example, if several students contribute "hyperthyroidism", then its confidence might be 60 percent. In this case, if an expert agrees that 'hyperthyroidism is a valid hypothesis for students, than the expert has confirmed an entry that already had high confidence. Thus, we hope to see that experts generally agree with high confidence entries and disagree with low confidence ones. This would imply that students are generally contributing accurate knowledge. Table 5.10 confirms that this is, in fact, the case.

	Verified EEKB Items	***Confidence of Confirmed Entries***	***Confidence of Denied Entries***
Avg:	0.810	52.856 %	23.238 %
Median:		38.500 %	16.000 %

Table 5.10: Results of analyzing KRG expert responses when asked to verify the quality of specific EEKB entries.

We observe that 81 percent of the generated EEKB is confirmed by our KRG participants. This means that of our amalgamated EEKB entries, 81 percent of them were deemed suitable to be included in the knowledge base. This is consistent with our previous findings, and provides a cross-referenced value for precision of the EEKB. Additionally, we see that denied EEKB entries have an average confidence value of 23.238 percent (16 percent median) while confirmed EEKB entries have an average confidence of 52.856 percent (38.50 percent median). Recall that confidence defaults to 10 percent when an entry is added to the EEKB, and increases by 5 percent on each successive repeated addition of the entry by a different student. This provides further evidence of our algorithm's quality because lower confidence entries tended to be less 'true' than higher confidence entries.

This dataset also provides the opportunity to explore the presence of student misconceptions. We analyzed our data for occurrences of EEKB entries that were marked with a larger than 50 percent confidence value by our original algorithm, but were later denied by KRG users. This represents entries for which many students agree, but does not turn out to be accurate. We consider these cases to all be examples of misconceptions. Table 5.11 below summarizes the misconceptions found.

Total Verify Questions Asked	***> 50 % Confidence And Denied***	***Misconception Rate***
420	5	0.0119

Table 5.11: Summary of misconceptions discovered in EEKB, e.g., hypotheses considered true by students and false by experts.

We see that the misconception rate for our EEKB is low, yielding only five detected misconceptions among 420 relevant opportunities. This yields a misconception rate 1.19 percent.

5.5 KRG Analysis Results

The previous section used the results of our knowledge-refinement game (KRG) dataset to validate the quality of the originally generated EEKB. We now present results of how well the KRG usage improved and refined this EEKB. This study was repeated due to poor initial results. Thus, we first present the negative results from the initial study. We briefly describe the system changes that were made in turn and present results from the secondary study. We find that problems with the design of the KRG grossly limited its efficacy and led to apparent losses in EEKB quality in our initial study. Many of these

losses can be traced back to some subtle but fatal design flaws that are discussed in detail in Chapter 6. Our results are greatly improved when errant factors from the study are improved.

Figure 5.6 below show EEKB recall over time as students used the KRG (study 1). We see that for the most part, EEKB recall remains steady while dropping off a couple of percentage points.

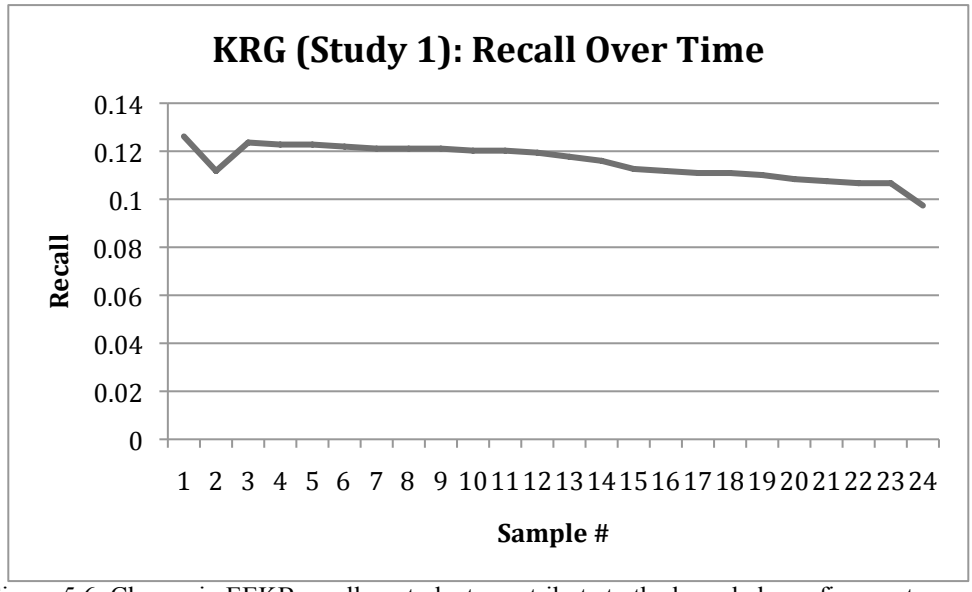


Figure 5.6: Change in EEKB recall as students contribute to the knowledge-refinement game.

It appears that somehow useful data is being removed from our knowledge base when our experts contribute. Figure 5.7 shows how precision changes over time. We see that precision falls even more starkly until the final phase of KRG usage.

We see that, again, somehow our KRG users are removing nodes that are believed to be correct from the EEKB. It's important to note the increase in precision at the end of

Figure 5.7. This occurred during the combining phase, in which the KRG found similar nodes and asked students if they should be combined. It appears that this technique is the one aspect of Dr. Doctor that led to improved EEKB quality.

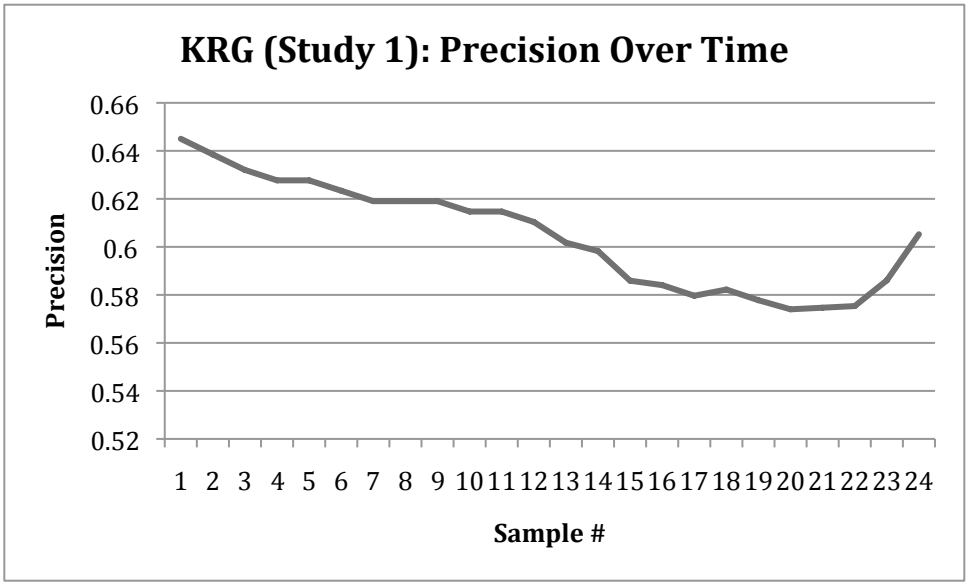


Figure 5.7: Precision over time as users contribute to the KRG

We decided to analyze the data further in order to find potential reasons for this lack of improved EEKB quality. Throughout most of the KRG sessions, experts were asked to submit pithy alternatives to the student generated hypotheses or data. We found that several participants misunderstood this process and instead removed essential information from the node in question. For example, in one case the original student data was:

"Do you have any allergies? No"

When asked to provide a shorter alternative to this data, one expert replaced the contents with the following:

"No"

This example illuminates the expert's misunderstanding that the given information had to contain the question and not just the answer to it. Because the data for this node was changed based on the expert's response, the node would no longer be considered correct by our analysis and would hurt both the recall and precision of our EEKB.

To determine the potential impact of this issue, we examined the KRG log for instances of "broken entries" in which experts accidentally worsened the EEKB. The results of this analysis are shown below in Table 5.12:

EEKB Nodes	***EEKB Relations***	***Total***	***Num Broken Entries***	***Percent Precision***
124	66	190	10	**5.26**

Table 5.12: Expert's contributed bad alternative data that results in a five percent decrease in EEKB precision

Thus, this accounts for a significant percentage of our decrease in precision as shown in Figure 5.7. To validate this and other issues, we removed question types from Dr. Doctor that did not require yes/no responses. This removed the potential for accidental broken entries introduced by experts. Three of the participants agreed to play Dr. Doctor a second time and we requested that the players contribute 100 additional answers. As before, the participant with the highest score received a gift. After one additional week of

the game being available, two of the three participants contributed vastly more than asked, answering 250 and 235 questions respectively. The third participant answered just above the minimum required, logging 105 answered questions.

We then re-ran our analysis for precision and recall of the EEKB based on improvements made by experts, incorporating only data from the revised study. As before, we captured a snapshot of the EEKB after every 15 KRG contributions. We then simply calculated the EEKB precision and recall for each snapshot to analyze the change in quality over time. To stay consistent with all of our analyses from this document, we calculated these statistics for an accepted knowledge base[15], which only includes entries that exceed a consistent threshold. Thus, it is possible for the EEKB size to grow as the KRG is utilized. Results for the change in EEKB precision over time given our second group of experimental data are shown in Figure 5.8.

We found that as experts played the revised Dr. Doctor game, the precision of the knowledge rose by four percent. We also re-measured knowledge *recall.* A display of the change in recall over time can be seen in Figure 5.9.

[15] *By "Accepted Knowledge Base", we mean an EEKB that only includes entries whose confidence values exceeed a given threshold. For our experiments, this threshold was set to 30 percent confidence, which represents at least 5 students being in agreement.*

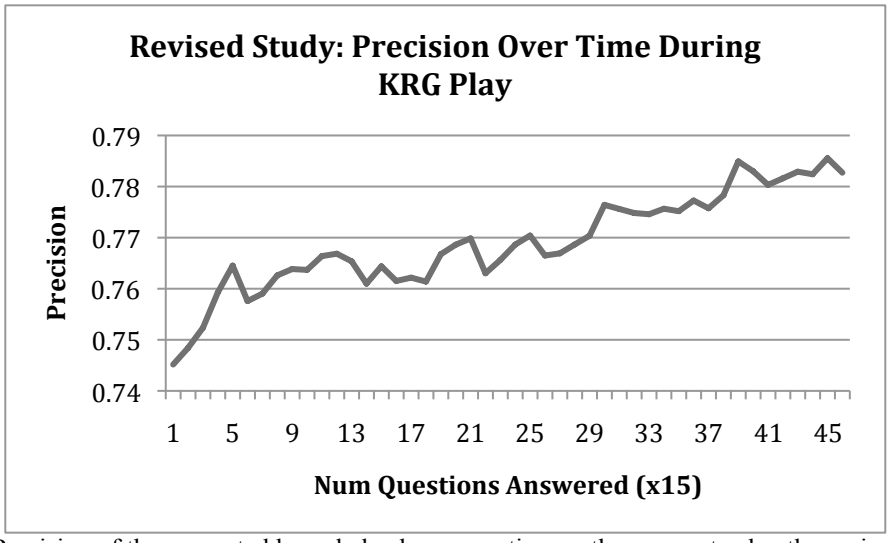


Figure 5.8: Precision of the generated knowledge base over time as three experts play the revised version of Dr. Doctor

We observed a nine percent increase in knowledge recall as our participants played Dr. Doctor. This occurred because players were able to effectively confirm knowledge entries that had low confidence and were previously excluded from the knowledge base quality calculation.

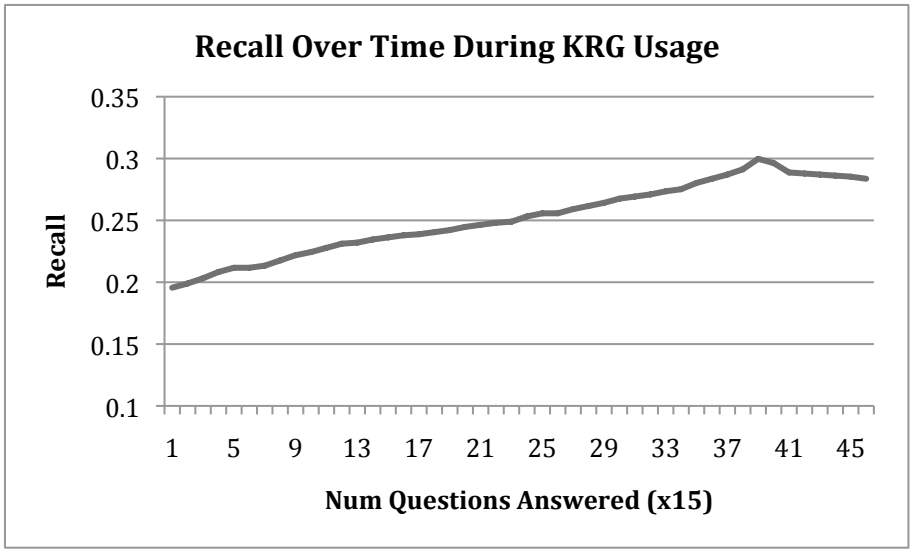


Figure 5.9: Recall over time. KRG play leads to a larger breadth EEKB once experts used the revised KRG that contained only Yes/No answers.

5.6 Conclusion

In conclusion, our data shows very promising results. We see that by utilizing our knowledge base acquisition algorithm, we can obtain knowledge bases with acceptably high precision (greater than 60 percent for all cases considered). We also find that these EEKB models are relatively small due to the limit in breadth explored by typical students who are contributed the information. However, this leads to the result that an EEKB can be fully saturated in merely one hour if 300 students are able to use Rashi in parallel.

We also find evidence that game mechanics play a role in enhancing our approach. We found that game mechanics in Rashi lead students to contribute more information, which helps build the EEKB model more efficiently. In addition, our knowledge refinement game was an effective way to validate the quality of our EEKB and shows that the presence of student misconceptions was trivial.

Our initial usage of the KRG did not lead to improvements in EEKB quality. We suspected that a few simple design flaws led to this result. In particular, when asked to contribute alternative data for specific EEKB nodes, experts often provided junk data by accident, leading to a lower quality EEKB. By simply removing this feature and re-running our experiment, we found that KRG game play lead to a four percent increase in precision along with a nine percent increase in recall.

In the next chapter, we discuss the implication of these results and what can be done for future work to further our goal of building quality knowledge bases automatically.

CHAPTER 6

IMPLICATIONS OF AUTOMATIC EEKB GENERATION

The results presented in chapter 5 illuminate the possibility of developing intelligent tutoring systems (ITS) that adapt and learn by coalescing the input of student users. In fact, most ITS that currently exist employ very few artificial intelligence techniques in practical environments [103]. This does not mean to imply that ITS do not contain intelligent features [7, 34, 62], but rather that most current systems do not learn over multiple student sessions. Our work aims to provide techniques and processes for deploying educational technologies that do more than individualize a tutor to a single student over a short learning session. Rather, our approach is to work towards intelligent tutors that are able to learn from collections of students over multiple classes, semesters, and years in an effort to individualize student experiences by understanding the student experience as a whole.

The results presented in the previous chapter provide evidence that our approach is both viable and efficient. This chapter provides proper contextual interpretation of the experiments and their results described in this document thus far. The widespread use of expert knowledge bases is mostly hindered by their extensive build times. Authoring tools have provided technological methods for building such models more quickly [65] but have not been utilized extensively for building domain level models. Our approach differs in that we attempt to obtain a model by analyzing student tutor usage. This approach is unique because the students are learning from the tutor regardless, and thus to

fully make use of their efforts, it is sensible to try to understand and organize the information that students provide.

This chapter begins by enumerating the hypothesis from chapter 3 and discussing whether the evidence from chapter 4 enables us to affirm or reject each hypothesis. We then make an argument for the deployment of Evolving Expert Knowledge Bases (EEKB). We discuss the advantages of such a model and the efficiency with which it can be created. We also discuss the learning gains that become possible by utilizing such models. The highlight of this chapter is then the presentation of a process for incorporating an EEKB into a generic project. The algorithms discussed in this document were designed to be domain independent and thus can be utilized by a vast array of tutoring technologies. We discuss, in detail, how an EEKB can be incorporated into a generic project and discuss the issues relevant to doing so.

We then discuss some of our peripheral findings that aim to support and enhance this process. In particular, we found that gaming mechanics in Rashi lead to higher quantities of student input. We discuss, specifically, how gaming mechanics and technologies can be utilized to increase the efficiency of EEKB generation and provide a more deep intrinsic learning experience for students. We also explore the results regarding our knowledge refinement game (KRG). In particular, we found little evidence in our initial study that using the KRG led to a more robust EEKB. However, many factors involved in the KRG and its use may have impacted this result. We discuss the lessons learned from this design and deployment process, and how small adaptations to a KRG led to

positive results. Our efforts suggest that the theory of knowledge refinement games remains sound and their practical applications remain promising. Thus we discuss general factors regarding how KRG design can be improved in order to maximize their efficacy.

We also discuss the opportunities for generalizing our approach and applying it to systems in other domains. In particular, systems with characteristics similar to Rashi can incorporate EEKB elements quite easily. Some of these characteristics include, but are not limited to, exploratory features, inquiry learning, and argumentation graphs. Additionally, if the domain in question can be modeled as sets of nodes and the relationships among them, then the incorporation of our approach becomes straightforward. We discuss, later in this chapter, how EEKB approaches can be applied to domains such as law, art history, and geology.

We begin, then, by arguing for the application of Evolving Expert Knowledge Bases across multiple domains.

6.1 Analysis of Results

In this section, we enumerate our hypothesis and reflect on the evidence provided for or against each. In general, we see that our experimentation yielded mostly positive results. The major exception to this is the usage of the knowledge refinement game, which in an initial study produced little effect on the quality of the EEKB, but in a follow-up study produced improvements in EEKB quality. We briefly discuss some of the possible

shortcomings of the KRG in the respected sections below. In addition, we attempt to contextualize these interpretations as much as possible, giving due respect to the need for further experimentation in different domains and practical situations.

6.1.1 Hypothesis 1: EEKB Precision

> ***Hypothesis 1:*** *Knowledge acquisition algorithms can obtain greater than 60% precision towards determination of the knowledge in an expert system within an ill-defined domain.*

We observed that when only considering the nodes of the EEKB, measured precision was consistently above 80 percent. This of course implies that the concepts that students explored within Rashi were generally well chosen. In addition, the measured precision of the entire EEKB was consistently above 60 percent for all cases considered. Thus, it appears that the provided experimental evidence supports hypothesis one. EEKB relationships seem to be less precise than nodes, bringing down the average precision noticeably. Several potential reasons for this exist. Namely, a large percentage of student relationships contain the default value of "neutral". Anecdotally, we observe that students mostly intended to express supporting or refuting relationships but forgot to change the value of the relationship within Rashi. This problem can be relieved by scaffolding, and by designing the Rashi software to make the requirement of this step more apparent. In addition, because of the exponential number of potential relationships between nodes, it becomes decreasingly likely that students will establish the same relationships that our human expert defined in the HEKB. We know that our HEKB does

not define all relevant relationships, a minor drawback to the methodology presented in this paper.

Additionally, it is important to note the potential uniqueness of the evaluation results for this particular tutor. All of our test cases were in a specific domain (patient diagnosis) and the cases were largely similar to one another. We did obtain a nicely randomized set of students in terms of age, experience, gender, and background but cannot necessarily claim that our results will transfer smoothly to other domains. Our intuition is that the results in other domains will remain strong, but we acknowledge that domain level factors (nomenclature for example) may impact our algorithms ability to coalesce student knowledge effectively. This motivates a desire to test our approach within Rashi, but for another domain significantly different than medical diagnosis, e.g., forestry or geology.

We must also acknowledge the possible effect of case difficulty on our results. Our four cases are generally the same level of difficulty and thus little randomization on this variable occurred. It is possible that increasingly difficult cases would lead to more erratic and incorrect student behavior. We believe however that this is not the case in practice. We believe that difficult cases would still leave students exploring valid topics of inquiry, even if those topics are misguided. We also posit, generally, that the application of lessons from Intelligent Tutoring System design [103] would alleviate any potential issues considerably.

We discuss future efforts that might dispel these concerns in our future work section below. Despite these potential issues, we intuitively believe that our algorithm is generally effective at producing precise EEKB models, and the evidence we have collected supports this notion. The results of our studies are thus in support of our first hypothesis.

6.1.2 Hypothesis 2: EEKB Recall

> ***Hypothesis 2:*** *Knowledge acquisition algorithm can obtain greater than 50% recall on average towards the identification of knowledge in an expert system within an ill-defined domain.*

At first glance, it might appear that our algorithm was unsuccessful in producing an EEKB with acceptably high recall. Our results show less than 20 percent measured recall in all four cases. However, what our results instead reveal about our students is that they tend to exhibit focused inquiry behavior, wavering little from promising avenues of search. Student work in Rashi focuses on a select subset of topics that are relevant and well informed for each particular case. This intuition is supported by previous studies in which student chat detection was done automatically, and it was ascertained that students only discuss 10 to 20 percent of our human knowledge base material [109].

In addition, our initial effort to create a human constructed knowledge base (HEKB) was focused on defining all possible student paths of exploration, and not necessarily the most likely ones. Thus, we seem to have learned from our analysis here that much of our

human generated knowledge base may have been constructed in vain, and the process could have been done with greater efficiency. Of course, it is hard to tell which 20 percent of the domain topics students are most likely to focus on, and this is another compelling argument for using an EEKB over an HEKB. The resulting EEKB model is reflective of domain topics provided by the student body that used the tutor.

Thus, we cannot claim that our hypothesis as stated was supported by our data. However, it appears that the EEKB correctly inferred the fraction of the knowledge with which students are generally engaged. This seems true because the data set analyzed is the same as in previous studies [109]. These studies show how students only explore about 20 percent of our knowledge base on average, and how intelligent coaching can provide more meaningful suggestions by searching this subset of knowledge when generating feedback. It is still likely that a more finely crafted and focused human knowledge base would have yielded a better recall from the EEKB. However, we conclude that the recall of our generated EEKB models is sufficient and useful in a practical setting.

6.1.3 Hypothesis 3: Efficiency of EEKB Construction

> ***Hypothesis 3:*** *A knowledge base can be built in significantly less time using our knowledge acquisition algorithms as compared with subject matter experts.*

The support for hypotheses one and two provide evidence that both automatic and human created knowledge bases are accurate. However, judging the efficiency of constructing a

knowledge base (whether it is done automatically or by hand) becomes difficult after considering some of the key variables in question. Our results show that the amount of construction done by a single student versus a single human expert is comparable (even though we would expect the expert to be slightly more efficient). We found that approximately 300 students can saturate an EEKB in merely one hour. Thus, if all those students are available at one time, the EEKB can be constructed very quickly. In addition, our analysis is slightly skewed in that the human constructed knowledge base is notably larger than the algorithmically constructed one. However, normalizing our human experts creation time for size still yields an advantage for implementing EEKB autonomous construction.

The strongest efficiency advantage arises because students outnumber teachers. We assume that the opportunity for parallelization exists equally within both groups. However, it is well known that in most academic settings, students do indeed outnumber teachers significantly[16]. Thus we can strongly conclude that under the assumption that available students outnumber available teachers at somewhere close to a 15:1 ratio, constructing an EEKB using our approach is significantly more efficient. In addition to this, neither students nor teachers are asked to perform actions outside of their normal interactions. Students contribute to the knowledge base while simultaneously participating in a typical learning setting. Thus, we conclude that our approach is more beneficial than having experts construct knowledge bases by hand, even if doing so via an authoring tool.

---

[16] *http://nces.ed.gov/fastfacts/display.asp?id=28 provides statistics on student to teacher ratios. In public schools in 2009, the student to teacher ratio was 15.4*

6.1.4 Hypothesis 4: Effect of Gaming Mechanics Within an ITS

> ***Hypothesis 4:*** *Gaming mechanics in an intelligent tutor (such as the patient treatment panel) lead to a higher quantity and quality of student input.*

This hypothesis was intended to showcase basic effects that game mechanics can have on a student's usage of an ITS. Game mechanics have been shown to lead to increased measures of presence along with heightened student engagement and motivation [58, 110]. Thus, it is sensible to infer that subtle aspects of ITS design might affect students' ability to contribute to an EEKB.

We began by utilizing a very simple measure, the quantity of student input, between two experimental groups. We discovered that the group with additional game mechanics within Rashi produced significantly more data that could be utilized by our EEKB algorithm. Thus, these features can be utilized to help students work and learn with an ITS more effectively while also contributing more efficiently to generated expert models. It is important to note that this initial experiment only observes an increase in the quantity of input with no regard to its quality.

We then examined this hypothesis further by estimating grades for each student in the respective experimental groups. We were able to show that the student group provided with game mechanics not only contributed more information, but was also rated as

having higher quality[17] information in their arguments. This is important because it helps reduce the evidence that games engage students more than do tutors without game mechanisms and thus students to enter more information, but that they do so with less accuracy. We found this difference to be statistically significant.

Also, there are additional ways in which Rashi may be able to increase this effect. The current Rashi interface is quickly becoming outdated, and the tools and interaction components feel old and rustic. An updated user interface along with increasingly stimulating interactions may help students contribute even more effectively. In addition, we plan to implement a final diagnosis component in Rashi that allows students to submit their final decision to the patient and provide a treatment plan. Student's ability to perform this step well will help them earn points or promotions. Although these features would likely be beneficial, we do find that even the simple measures taken in the work described here proved effective, leading to both a significant increase in the quantity and quality of student work. Thus, we find strong support in favor of hypothesis 4.

6.1.5 Hypotheses 5 & 6: KRG Effects on EEKB Precision and Recall

> ***Hypothesis 5:*** *A Knowledge Refinement Game used by domain experts leads to 5-10% increase in precision of the evolving expert knowledge base created by students.*
>
> ***Hypothesis 6:*** *A Knowledge Refinement Game used by domain experts leads to 5-10% increase in recall of the evolving expert knowledge base created by students.*

[17] *Please see Chapter 4 for the definition of "higher quality" Rashi work. This is essentially our best estimate of the depth, breadth, and quality of a student's argument on a Rashi case.*

The evidence for these hypotheses provided mixed results, and seem to be highly dependent on the configuration of the KRG. We found in our initial experimentation that expert's playing our knowledge refinement game led to lower quality in the final EEKB model. This result is unfortunate in that it provides evidence against the utility of such games, but also opens potential avenues of further exploration. We do not see any evidence that our KRG players were incapable of correctly judging the quality of EEKB entries. However, we did observe a noticeable quantity of entries in which experts misinterpreted their task only to accidentally lower the quality of an entry. We called these "broken entries" in which the player inadvertently removed the most pertinent information. These "broken entries" account for a significant percentage of the decreased EEKB quality (see chapter 5). When the features that led to these broken entries was removed, and our experiment re-issued, we found that KRG play led to a higher quality EEKB with respect to both of our metrics.

We do not believe that the nature of the removed question types within Dr. Doctor caused the EEKB models to decline in quality, but rather that the design of these features did not cultivate expert responses that took full advantage of their intention. A more carefully designed KRG could likely incorporate direct manipulation of node and relationship annotations, but would need to do so while carefully ensuring that any annotation change is not likely to lead to broken entries.

Thus, we conclude that the evidence collected provides a weak support of hypotheses five and six. We find evidence that having many experts play a knowledge refinement game leads to increased EEKB quality. However, this finding is highly dependent on the features of the KRG and how well they support players in providing quality feedback. For our studies, simplifying the interaction between the players and the KRG vastly improved the quality of the feedback, and thus the EEKB models.

6.1.7 Hypothesis 7: EEKB Misconception Rate

> ***Hypothesis 7:*** *Less than 5% of acquired student data is considered a mass misconception (for human biology Rashi use) as identified by a Knowledge Refinement Game.*

When utilizing student input (rather than expert input) for generating models, it is natural to assume that student misconceptions may be interpreted by the model as true knowledge shared by many peers. The knowledge refinement game provided evidence for benefitting the EEKB and it also proved to be an effective tool for studying potential misconceptions. Experts used the KRG and rated student contributions as accurate knowledge or not. We determined that topics in the EEKB that both were high in confidence and were deemed inaccurate by experts could be interpreted as misconceptions. In practice, we found a very small presence of misconceptions (1.19 percent).

Thus, the EEKB is sufficiently void of misconceptions, providing evidence that student contributions are generally accurate. This is one hypothesis that seems more susceptible to variation based on the domain. It is possible that other domains may reveal higher misconception rates. However, we do find, at least in medical diagnosis, evidence that EEKB generation can be accomplished with a low misconception rate.

6.1.8 Conclusion

We thus conclude that the majority of our hypotheses have been given support from the evidence we have collected. We find that constructing an EEKB from student ITS data is efficient and that it can be produce a precise and suitably large knowledge base. In addition, we see that gaming mechanics within the source ITS can lead to improved student behavior and more efficient EEKB creation. We do find evidence that the knowledge refinement game provides a benefit to the quality of the EEKB, however this result is dependent on the KRG fostering quality expert feedback.

Table 6.1 summarizes the results of our analysis for this dissertation. This table provides a simplified view of our results by listing our hypotheses and simply displaying whether we found weak/strong support or refutation for each. We see that all of our hypotheses are supported, some more convincingly than others.

Hypothesis 1: EEKB Precision	*Strong Support*
Hypothesis 2: EEKB Recall	*Support*
Hypothesis 3: Efficiency of EEKB Construction	*Strong Support*
Hypothesis 4: Effects of Gaming Mechanics	*Strong Support*
Hypothesis 5: KRG Effect on Precision	*Weak Support*
Hypothesis 6: KRG Effect on Recall	*Weak Support*
Hypothesis 7: EEKB Misconception Rate	*Support*

Table 6.1: A simplified summary of our hypotheses and whether our experimentation supports or refutes them

We continue this chapter by discussing why it is sensible for ITS designers to incorporate EEKB generation technology into their systems. We then discuss a process for doing so, and then conclude the chapter with a discussion of the necessary future work.

6.2 Why Build an EEKB for an Intelligent Tutor?

We believe that there are three broad reasons why EEKB models should be a viable consideration for designers of intelligent tutoring systems. Firstly, they provide the domain-level model for enabling individualized coaching. Once an EEKB is developed, intelligent algorithms can be applied that utilize this model to provide students with individualized feedback. Experiments show that this coaching can lead to learning gains and increasingly positive student behavior [33]. This semantic coaching works by performing the following steps:

> ***Detection:*** *An intelligent coach that is monitoring a student's actions detects the current focus of student work. This can be done in a variety of ways. Rashi does this by analyzing various textual inputs that students provide. This input is then matched to a particular concept within the EEKB. Detection thus allows the tutor to understand various aspects of a student's focus. Namely, the tutor can understand concepts that are (or previously were) being explored by the student, as well as the concepts in which students have yet to perform.*
>
> ***Determining Optimal Related Concepts:*** *The EEKB relationships are then useful for determining related concepts most near the student's current focus. For example, if a student is studying the possibility that a*

patient is ill with hyperthyroidism, then the coach can use the EEKB and its relationships to determine that particular blood tests are related to this concept. The coaching module can then determine that these concepts are the optimal concept for the student's later exploration.

Present Coaching to Students: *Once the previous two steps have been achieved, the tutor can present advice to the student, attempting to guide their behavior towards an optimal path. This can take many forms, and lessons in the field of HCI are surely applicable in how to present students with coaching suggestions. For example, one design decision is whether to actively engage (e.g., provide a popup window that interrupts students but ensures that they see the suggestion) or to passively suggest (e.g., provide a side panel that alerts students that advice is available without forcing them to acknowledge) this information to students.*

The second reason why an EEKB is valuable is that it supports coaching in a similar fashion but within a collaborative setting. In this situation, the coach detects the focus of multiple students' work. The second phase of coaching (to determine optimal related concepts) can be accomplished by exploring not only the semantic relations within the EEKB, but by also by considering the efforts of other students. This is an attempt to recognize situations in which pairs or small groups of students might benefit from collaborating. Thus, stage three above takes the form of suggesting to students that collaboration might be a strong choice of activity.

The third advantage of using an EEKB is that the model is naturally reflective of the student body that uses the related tutoring system. The EEKB begins as an empty model (0 nodes and 0 relationships) and grows in size as students contribute knowledge. This

knowledge that eventually fills the EEKB model is provided by the student body themselves. Thus, the EEKB is naturally relevant to the students (and those with similar characteristics) that contributed the knowledge.

The final argument for incorporating an EEKB is the ease with which it can be done. Deployment is domain independent, and thus can be a potential fit for any domain (see section 6.1.1 for an evaluation of various domains and relevant EEKB applicability). Our EEKB module (currently available only in Java) can be plugged into any tutoring system that supports Java or communication with Java. Developers will be able to use this module to construct EEKB models, save/load them to files, and even use a coaching module to provide dynamic feedback to students. A thorough description of the process of incorporating EEKB models into any project is described in section 6.2.7.

We now describe, in detail, the process for incorporating EEKB models into Intelligent Tutoring Systems that instruct in various domains.

6.3 A Process for Incorporating EEKB Intelligence into any Project

Incorporating an EEKB into an Intelligent Tutoring System is simple and requires very little configuration. The only true requirement for EEKB models to be effective in a particular domain is to establish how the inherent graph structure fits that domain. As long as this mapping can be established, then incorporating an EEKB into a project becomes straightforward.

In this way, EEKB incorporation is fully generalizeable. We simply download the EEKB generation module and incorporate it into an existing tutor, invoking relevant function calls. The summary of this process can be seen in Table 6.2. The configuration stage is the most relevant because users must establish what node types (hypotheses, data, etc) and relationship types (supports, refutes, etc.) are valid for the domain in question. This must be configured manually for each particular tutoring system. Once this is done, the user must also add relevant function invocations throughout the tutor's code. This is represented step 3 in Table 6.2.

Step Number	***Step Title***	***Description***
1	*Download*	*Download the EEKB generation Java module*
2	*Configure*	*In the module, establish node types; establish relationship types;*
3	*Hook up*	*Invoke nodeEvidence(), relationshipEvidence(), saveEEKB(), loadEEKB(), etc. in your tutor.*
4	Use the tutor	Allow students to use your tutoring system normally.
5	*Refinement (Optional)*	*Use tools or knowledge refinement games to manually improve EEKB.*
6	Coach	Coach in the tutor uses EEKB to provide dynamic feedback to students.

Table 6.2: Summary of process steps for incorporating EEKB into ITS project

We note that step four is the one step that developers of tutoring systems perform regardless of whether EEKB modules are incorporated. We now move through the steps in succession and describe, in detail, how to perform each step to set up an EEKB in any ITS project.

6.3.1 Download

This is the simplest of the steps. The code must be downloaded. The EEKB module is currently only available in Java[18]. It is advised that this module be incorporated as server side code, so that several students can contribute to one single EEKB. The code can be found at:

http://althea.cs.umass.edu/EEKB/EEKB.zip

Simply download this package, and incorporate the unzipped code into your project. The source code is provided so that necessary changes can be made.

6.3.2 Configure

Our EEKB module requires minimum configuration. The main aspect of the code that must be configured is the set of node types along with the set of relationship types. These can vary for different domains, and so it is important to choose these wisely. For our efforts in patient diagnosis, we chose the node types as follows:

NodeTypes *= { "Hypothesis", "Data", "Inference" }*

You must also choose relationship types. This should be the total enumeration of all possible relationships between two nodes. For our purposes in patient diagnosis, we chose to use:

[18] *www.java.com*

RelationTypes *= { "Weakly Supports", "Supports", "Weakly Refutes", "Refutes", "Is Consistent With", "Is Not Consistent With", "Not Related To" }*

This is the only necessary configuration. You are ready to add the EEKB code to your project by invoking some of its functionality.

6.3.3 Add Method Invocations

The next step is to add function invocations that create, modify, save, and load an EEKB within your project. There are five main pieces of code that must be added:

> ***Create an EEKB:*** *When your code loads up, it should create a new instance of an EEKB or load an existing EEKB from a save file (see LoadEEKB below). You can create an EEKB by simply creating a new instance of the EvolvingEKB object*[19].
>
> ***Invoke NodeEvidence() and RelationshipEvidence():*** *You must invoke these methods everywhere in your code that a student provides evidence that a certain topic or relationship may exist in your domain. For example, if a student has to type in a hypothesis somewhere, you might take their entry and pass it into the NodeEvidence() invocation. The EEKB takes care of the rest!*
>
> ***Invoke SaveEEKB():*** *You must invoke this function to ensure that your EEKB is saved to a file and not lost when your tutor shuts down. Simply ensure that this function is invoked on your EEKB early and often.*

[19] *See* ***EvolvingEKB.java*** *from within the code base for more information*

Invoke LoadEEKB(): *This function allows you to recover a saved EEKB from a file when necessary. This is most useful when restarting the application.*

That is it! Once these invocations are in place, your system is ready to automatically construct an EEKB model directly from your student's input.

6.3.4 Have Students use Tutor

This step is simply because it is necessary whether an EEKB is incorporated into the project or not. Simply have students use the software normally. As students work, the software will send their data into the node and relationship evidence methods and the EEKB model will be constructed automatically.

After this step, authors can optionally choose to refine the knowledge in any way they see fit. We do not provide details here of any refinement suggestions and do not currently have any plans to make our knowledge refinement game available for public use.

6.3.5 Refine the Knowledge

Once an EEKB model is generated automatically, it is important to refine the model in some way to ensure increasing robustness. We suggest doing so automatically by incorporating a system such as a knowledge-refinement game. If employed correctly, knowledge refinement games can provide a venue for simply submitting knowledge bases for automatic improvements. A well-designed KRG, that is domain independent can accept an EEKB as input, and employ players across the Internet to garner necessary

refinements. The result is a model with added value that can be applied to improve student learning.

We acknowledge that other approaches are available. Knowledge refinement is a more reasonable task than knowledge engineering, and thus in some situations performing this task by hand may be appropriate. However, this approach is certainly not scalable. Thus, automated approaches such as knowledge refinement games will provide a venue for quickly improving EEKB models of scaling size and complexity.

6.3.6 Coach Future Students

Once the EEKB is constructed and refined, it can be used to coach future students to improve their learning within an ITS. This involves using the coach to detect the topic of current student focus, and then utilizing the semantic relations of the EEKB to offer dynamic feedback. Our coaching module currently performs these tasks already. Those interested also have the option of implementing their own coaching module. See section 6.2 above for the high level overview on how the coaching module works. For more detail, see [117] for a strong resource on coaching in ill-defined domains.

6.3.7 What Fields will our Approach Work For?

EEKB models were originally designed for Rashi cases (specifically those in human biology), but are easily extendable to other domains. In fact, because Rashi is a domain-independent tutoring system, the EEKB naturally transfers to other domains quite easily. In general though, EEKB models match most perfectly with domains that can be

described as ill-defined [2, 3]. This means that problems within that domain generally do not have a purely correct or incorrect answer, but rather are based on the strength of a students understanding of conceptual ideas.

EEKB models generally work best in ill-defined domains because these domains can usually be described succinctly as sets of nodes with relationships between them. Below are some examples of domains for which our approach should work well without any major complications:

> ***Law:*** *The law is naturally ill-defined and can be represented as a set of court cases, results, and bills (among others) along with the relationships between these topics. The relationship set could include semantic relationships such as supports, gives precedent to, etc. This EEKB would be useful for assessing a student's area of focus and quickly discovering related items that are central to his or her goal. Some work on tutors in the domain of law can be found in [1][8].*
>
> ***Art History:*** *The history of art is another ill-defined space that could benefit from an EEKB [33]. This domain can be represented as a set of features (e.g., impressionist), artists, art pieces, etc. along with the relationships among those (Vincent Van Gogh created the Mona Lisa). Tutors in this domain could provide guided explorations through this information in a natural and optimal way.*
>
> ***Forestry:*** *This is an interesting domain that requires students to be able to analyze features of wooded areas to draw conclusions. For example, our Rashi case in forestry presents students with the results of a fire, and*

students must collect clues and apply knowledge from forestry to piece together the fire's cause.

Psychology: *Similar to the patient diagnosis described in this work, psychologists could benefit from cases involving virtual people with mental afflictions. The student would need to diagnose the patient effectively. An EEKB defining symptoms, various afflictions, and their relationships would prove to be an invaluable resource for this exercise.*

Ethics: *Ethics is another ill-defined domain that can benefit from EEKB. In general, the domain might be structured as a set of potential, ethical viewpoints, or value systems and their relationships to one another.*

There are many other fields for which our approach may prove useful including engineering, literature, history, or even poetry. However, it is important to note that not every domain is easily defined with an EEKB or similar model. In fact, many well defined fields such as mathematics and physics are so well defined that tutors may actually work best by utilizing other approaches [6, 15, 55, 56]. However, it is not inconceivable to imagine a system that constructs, say, domain level math hints by having students contribute their solutions to some kind of master knowledge base.

6.4 Future Work

This dissertation focused on the knowledge development stage in which students use an ITS and an EEKB is constructed from the data delivered during these interactions. Future work however will primarily focus on how the EEKB performs when scaffolding and guiding the behavior of future students. We are specifically interested in comparing

the quality of feedback given to students when the EEKB is utilized compared to the larger HEKB. Additionally, the input of future students can be used to continuously refine the EEKB data. We also hope to deliver EEKB models in practical settings for domains other than human biology.

The other aspect of our future endeavors will involve the improvement and re-testing of the knowledge refinement game. Many issues with the KRG design and implementation arose during our tests, and thus it is sensible to work to improve these flaws to judge whether a KRG is potentially useful for refining EEKB quality. We also hope to reissue the features that were removed from our more successful study in a way that leads to positive results.

6.4.1 Using the coach with the EEKB

Rashi's coaching module is already implemented and uses an expert knowledge base to reason about a student's optimal path through the software. We are interested in comparing the quality of feedback produced by a proposed coach when using a generated EEKB and compare this to similar feedback while using the HEKB. To measure this, we would probably employ feedback questions within Rashi that ask students to judge the quality of the feedback they are receiving. With two randomized groups, we could then compare the qualities of feedback between different backend models. We are not necessarily looking for the EEKB to outperform the HEKB in this case. It is sufficient for us to show that a generated EEKB performs equally to an HEKB because we have

shown in this dissertation that EEKB models can be built in vastly less time. Thus, equal performance would still be considered a strong result.

6.4.2 Continuing EEKB Refinement

Future work will also observe how EEKB refinement continues over multiple iterations of student usage. In particular, we hope to see the EEKB slowly and steadily gather increased precision and recall as additional students utilize it. In addition to this, we hope to implement in-time construction of an EEKB in Rashi. Currently, the data is poured into the software after students have used Rashi. This feature is a minor extension of the current framework, but would help to explore some additional issues. Namely, how well does an EEKB work if it is being amalgamated during student usage, and the coach is utilizing a nascent EEKB for its feedback? These issues and more can be explored in more depth once the EEKB is constructed in time.

We also note the presence of negative feedback loops, defined as pieces of information that are consistently contributed by student users, and then consistently removed by refinement technologies such as the KRG. This causes inefficiency in that the inaccurate information is contributed multiple times and must also be removed for each occurrence. Thus, simple extensions to our code base will ensure that inaccurate information is stored and not reinstated by future student contributions.

6.4.3 Extend to Other Domains

We hope to extend our efforts into other domains for several reasons. Firstly, providing evidence of EEKB quality in multiple domains helps strengthen the argument for their generic usage in ITS. In addition, we hope to provide evidence of the utility of our approach in multiple domains to encourage ITS designers to utilize EEKB models.

Rashi is already implemented as a domain independent inquiry tutor, and cases have been developed for several domains. These include art history, geology, forestry, and ethics. Thus, the path to EEKB development in these domains is straightforward. We merely need access to students to use these cases. If these students can be recruited, then we can begin to study how EEKB development occurs in domains other than patient diagnosis.

6.4.4 Establish Benchmark HEKB Models

One major problem with the development of EEKB models in other domains is our lack of HEKB models to which to compare them. Our human biology models developed by subject matter experts were sufficient for the studies within this dissertation, but most of our cases in other domains are lacking sufficiently robust HEKB models.

Thus, we believe it makes sense to develop HEKB models in any domain within we work as benchmark goals for judging the quality of future EEKB generation algorithms. In this way, competing improvements to our techniques presented here can be compared with a standardized method. We believe that if the field of ITS becomes serious about exploring

the processes presented in this dissertation, then benchmark knowledge bases will become key factors in improving the methodology.

6.4.5 Improving the Knowledge-Refinement Game

We observed negative results when analyzing KRG usage within the parameters of our initial study. We do not believe that this is due to a lack of sound theory. In fact, we believe that a relatively simply design problem led to these negative results, e.g., allowing experts to enter text phrases without understanding the requirements of the text phrases. Thus it is important for future work to focus on the potential improvements necessary to see positive (or improved) results from KRG usage. We do acknowledge that positive results were obtained when removing the features in question. However, we hope to find effective ways to incorporate the removed features.

Our successful design improvement was to give experts less control over the actual data stored within an EEKB entry and to eliminate the possibility of textual input into the KRG. Initially, we chose to allow experts to alter textual description of EEKB data directly in order to provide more clear and pithy data. Unfortunately, this led to many entries in which data became inadvertently useless and the original information was lost forever. This also led to decreases in our measures of EEKB quality. It is also important to note that the software potentially could have done more to prevent this effect. Perhaps the instructions could have been clearer, examples could have been provided, or basic bad responses could have been detected and avoided. One possibility that makes intuitive sense is to at least restrict an expert's direct connection to this (textual?) data and allow

changes only through propositions that must be confirmed by other experts before being accepted permanently.

Lastly, the KRG's game mechanics may not have succeeded in producing a fun atmosphere for casually contributing to the EEKB. In anecdotal conversations with the participant experts, we found that the game mechanics (although not detracting) did not really distract experts from the fact that they were answering long lists of questions. Additional features could make this more bearable.

6.5 Conclusion

In conclusion, this dissertation presents a novel approach to developing knowledge bases for use in intelligent tutoring systems. We present the concept of an evolving expert knowledge base (EEKB) that is constructed by observation of student work within an intelligent tutor called Rashi. We observe that the quality of this knowledge base is acceptably high and can be built with significantly more efficiency than by hand.

In addition, we explore the effects of game mechanics on various aspects of automatic knowledge base construction. Specifically, we find that game mechanics within an ITS lead students to contribute data that is both higher in quantity and quality, which in turn leads to more efficient EEKB construction. We also explore a novel type of game called a knowledge refinement game (KRG) in which expert game players refine EEKB entries. We find evidence that the KRG leads to an improved EEKB.

We conclude that it is sensible for developers of ITS to consider incorporating EEKB technology into current systems. By utilizing our modules, an ITS can obtain a domain model that can be used to scaffold and coach future students, leading to an overall increase in the efficacy of ITS technology.

APPENDIX A

GLOSSARY OF TERMS

The following definitions reflect the intended use of these terms as they are employed throughout this document. In addition, this section is intended to help clarify the subtle differences between distinct but related terms.

Artificial Intelligence (AI): an active area of research that seeks to create machines and algorithms that showcase human intelligence.

Artificial Intelligence in Education (AIED): an active area of research in which artificial intelligence techniques are applied to computer-based instruction.

Authoring Tool: An application that abstracts away the necessity for programming expertise, allowing the creation of content, models, or behavior for Intelligent Tutoring Systems.

Competency Model: a model within the Evidence Centered Design framework. Defines how the knowledge (or competencies) of a student is obtained.

Computer Based Instruction (CBI): the use of computer programs to teach or provide instruction within a particular domain.

Constraint Based Tutor (CBT): An ITS that offers models the constraints of the problem space without knowledge of acceptable solutions or solution paths.

Crowdsourcing: An active area of research and application in which large groups of anonymous contributors are invited to perform fractional portions of a larger task.

Domain Independent Tutor: An Intelligent Tutoring System that is generalized to instruct in a variety of domains.

EEKB Precision: A measurement of the degree to which an Evolving Expert Knowledge Base's content is true. Defined by the percentage of an EEKB's content that is also contained in a thorough Human Expert Knowledge Base for a particular domain.

EEKB Recall: A measurement of the breadth of an Evolving Expert Knowledge Base's true content. Defined by the percentage of a Human Expert Knowledge Base's content that a generated EEKB contains in a particular domain.

Evidence Centered Design (ECD): A framework for developing assessments based on models that define how evidence of student competence is gathered and interpreted.

Evidence Model: a model within the Evidence Centered Design framework. Defines how learner actions inform evidence of their knowledge.

Evolving Expert Knowledge Base (EEKB): An Expert Knowledge Base that adapts its content and structure dynamically as additional evidence of existing knowledge is presented. Thus an EEKB is and EKB with additional properties.

Expert System: A specific type of Artificial Intelligence that attempts to mimic the reasoning of a human specialist.

Expert Knowledge Base (EKB): A semantic representation of knowledge in a particular domain. A subject matter expert (SME) normally constructs this representation.

First Order Optimal Strategy (FOO): A game development term used to describe the most optimal sequence of actions for accomplishing the goals in a given game. A game can be considered 'broken' when this strategy is not one the designers of the game intended.

Games with a Purpose (GWAPS): An active area of research in which games are used to enlist players that provide information or perform otherwise tedious tasks.

Human Expert Knowledge Base (HEKB): An Expert Knowledge Base that was crafted by one or more subject matter experts, either via an Authoring Tool or by direct content creation methods.

Ill-Defined Problem Space: a problem space that lacks clarity in how solutions are explored, discovered, and assessed.

Intelligent Tutoring System (ITS): A CBI system that employs AI techniques to dynamically assess, scaffold, or adjust to an individual student.

Inquiry-Based Learning: An approach to learning that involves presenting realistic problems and scenarios along with a facilitator who guides the learning process as student opportunistically explore to develop knowledge or solutions.

Inquiry Phase: A characteristically independent stage of the inquiry learning process. These stages are no consistent among models but usually include such phases as 'Developing Hypothesis', and 'Gathering Data'.

Knowledge Base: see *Expert Knowledge Base.*

Knowledge Refinement Game (KRG): Any application that motivates users with gaming mechanics and accepts user input for the purpose of altering the content and structure of an Expert Knowledge Base.

Recommender System: An information filtering system that attempts to rate or predict user preferences for an item currently unfamiliar to the user.

Serious Games: An active area of research in which games are implemented whose purpose is anything other than pure entertainment.

Stealth Assessment: A form of assessment that features an absence of student interruptions. Assessments are performed by modeling and observing student actions while using inference to judge student competency.

Subject matter expert (SME): a person who has authoritative knowledge on a certain domain, and shares this knowledge in order to create an expert model.

Task Model: a model within the Evidence Centered Design framework. Defines tasks that relate to the skills that must be ascertained.

APPENDIX B

EEKB SOURCE CODE

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  /**
4:   * Created by IntelliJ IDEA.
5:   * User: mfloryan
6:   * Date: Jul 3, 2012
7:   * Time: 1:52:19 PM
8:   * To change this template use File | Settings | File Templates.
9:   */
10: public class EEKBItem {
11:
12:     //internal id of this item
13:     protected int id;
14:
15:     protected EEKBItem(int id){
16:
17:         this.id = id;
18:     }
19:
20:     public boolean isNode(){
21:         return this instanceof EEKBNode;
22:     }
23:
24:     public boolean isRelationship(){
25:         return this instanceof EEKBRelationship;
26:     }
27:
28:     public int getId(){
29:         return id;
30:     }
31:
32:     public void setId(int newId){
33:         this.id = newId;
34:     }
35: }
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  /**
4:   * Created by IntelliJ IDEA.
5:   * User: mfloryan
6:   * Date: May 22, 2012
7:   * Time: 11:57:00 AM
8:   * Our matcher uses this to return EEKBNodes as results along with their scores
9:   */
10: public class EEKBMatchResult {
11:
12:     //the node that contains this result
13:     private EEKBItem resultItem;
14:
15:     //the matching score of this node
16:     private float score;
17:
18:     //constructor
19:
20:     public EEKBMatchResult(EEKBItem resultItem, float score) {
21:         this.resultItem = resultItem;
22:         this.score = score;
23:     }
24:
25:     public String toString(){
26:         return resultItem.toString() + " with score: " + score;
27:     }
28:
29:
30:     public EEKBItem getResultItem() {
31:         return resultItem;
32:     }
33:
34:     public float getScore() {
35:         return score;
36:     }
37: }
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  import org.apache.log4j.Logger;
4:  import org.apache.lucene.analysis.Analyzer;
5:  import org.apache.lucene.analysis.WhitespaceAnalyzer;
6:  import org.apache.lucene.analysis.StopAnalyzer;
7:  import org.apache.lucene.analysis.standard.StandardAnalyzer;
8:  import org.apache.lucene.document.Document;
9:  import org.apache.lucene.document.Field;
10: import org.apache.lucene.queryParser.QueryParser;
11: import org.apache.lucene.queryParser.ParseException;
12: import org.apache.lucene.search.*;
13: import org.apache.lucene.store.Directory;
14: import org.apache.lucene.store.SimpleFSDirectory;
15: import org.apache.lucene.util.Version;
16: import org.apache.lucene.index.IndexWriter;
17:
18: import java.io.File;
19: import java.io.IOException;
20: import java.io.FileReader;
21: import java.util.ArrayList;
22: import java.util.StringTokenizer;
23: import java.util.List;
24: import java.util.Scanner;
25:
26: import edu.umass.ckc.rashi.ekb.IndexableProp;
27:
28: /**
29:  * Created by IntelliJ IDEA.
30:  * User: mfloryan
31:  * Date: May 20, 2012
32:  * Time: 2:20:50 PM
33:  * To change this template use File | Settings | File Templates.
34:  */
35: public class EEKBMatcher {
36:
37:     //path to where the index file should be built
38:     private static final String INDEX_FILE_PATH = "cachedResources/IndexFiles_EEKB";
39:
40:     //this is the minimum score threshold, only matches with a score higher than this value are returned
41:     private static final double MATCH_SCORE_MIN = 3.0;
42:
43:     //The analyzer that this matcher will use.  Choices include:
44:         //StandardAnalyzer
45:         //WhitespaceAnalyzer
46:         //StopAnalyzer
47:         //SnowballAnalyzer
48:     private Analyzer analyzer = new LuceneStopWordAnalyzer();
49:
50:
51:     //stop words that are not analyzed
52:     private List<String> stopWords;
53:
54:     //the index writer
55:     private IndexWriter indexWriter = null;
56:
57:     //variables used for searching
58:     private IndexSearcher searcher = null;
59:     private QueryParser parser = null;
60:
61:     //the knowledge base that we want to be able to search through
62:     private EvolvingEKB knowledgeBase = null;
63:
64:     //true iff  ahuman will confirm the quality of a match
65:     private boolean humanConfirmMode = false;
66:
67:
68:
69:     //Some static variables for scoring in human confirm mode
70:     private static int numCorrectMatches = 0;
71:     private static int numIncorrectMatches = 0;
72:     private static int numCorrectIgnores = 0;
73:     private static int numIncorrectIgnores = 0;
74:
75:
76:     //constructor
77:     protected EEKBMatcher(EvolvingEKB evolvingEKB){
78:
```

```
79:         //clear the index directory
80:         clearIndexDirectory();
81:
82:         this.knowledgeBase = evolvingEKB;
83:
84:         //setup the stop words
85:         stopWords = getDefaultStopWords();
86:
87:         rebuildIndex(knowledgeBase);
88:
89:     }
90:
91:     private void setupSearcherAndParser(){
92:         try{
93:             //setup the searcher and parser
94:             searcher = new IndexSearcher(new SimpleFSDirectory(new File(INDEX_FILE_PATH)));
95:             parser = new QueryParser(Version.LUCENE_30, "data", analyzer);
96:         }
97:         catch(Exception e){
98:             e.printStackTrace();
99:         }
100:    }
101:
102:
103:    protected void rebuildIndex(EvolvingEKB evolvingEKB){
104:
105:        //clear the index directory
106:        clearIndexDirectory();
107:
108:        //System.out.println("Rebuilding index");
109:        //System.out.println("EEKB Node size: " + evolvingEKB.getNodes().size());
110:        //System.out.println("EEKB Rel size: " + evolvingEKB.getRelations().size());
111:
112:        this.knowledgeBase = evolvingEKB;
113:
114:        try{
115:            indexWriter = new IndexWriter(new SimpleFSDirectory(new File(INDEX_FILE_PATH)), analyzer, IndexWriter.MaxFieldLength.UNLIMITED)
;
116:
117:            ArrayList<EEKBNode> nodes = knowledgeBase.getNodes();
118:            ArrayList<EEKBRelationship> relations = knowledgeBase.getRelations();
119:
120:            //for each node, create a document and index it!
121:            for(EEKBNode node: nodes){
122:                Document doc = new Document();
123:
124:                //strip all non-characters from the data field
125:                String data = stripNonCharacters(node.getData());
126:                data = this.stripStopWords(data);
127:
128:                doc.add(new Field("id", ""+node.getId(), Field.Store.YES, Field.Index.NOT_ANALYZED));
129:                doc.add(new Field("data", data, Field.Store.YES, Field.Index.ANALYZED));
130:                doc.add(new Field("type", "Node", Field.Store.YES, Field.Index.NOT_ANALYZED));
131:
132:                indexWriter.addDocument(doc);
133:            }
134:
135:            //for each relationship, we do the same
136:            for(EEKBRelationship rel: relations){
137:                Document doc = new Document();
138:
139:                //strip all non-characters from the data field
140:                String data = stripNonCharacters(rel.toString());
141:                data = this.stripStopWords(data);
142:
143:                doc.add(new Field("id", "" + rel.getId(), Field.Store.YES, Field.Index.NOT_ANALYZED));
144:                doc.add(new Field("data", data, Field.Store.YES, Field.Index.ANALYZED));
145:                //doc.add(new Field("fromNodeData", rel.getFromNode().getData(), Field.Store.YES, Field.Index.ANALYZED));
146:                //doc.add(new Field("toNodeData", rel.getToNode().getData(), Field.Store.YES, Field.Index.ANALYZED));
147:                doc.add(new Field("type", "Relation", Field.Store.YES, Field.Index.NOT_ANALYZED));
148:
149:                indexWriter.addDocument(doc);
150:            }
151:
152:
153:            indexWriter.close();
154:        }
155:        catch(Exception e){
```

```
156:             e.printStackTrace();
157:         }
158:     }
159:
160:
161:     /*
162:     *    Searches the EEKB for a match
163:     *       @param searchType: the type of object that is required (e.g. "Node" or "Relation")
164:     *       @param confirmMathes: whether or not the matches should be verified by a human
165:     * */
166:     protected ArrayList<EEKBMatchResult> performSearch(String queryString, String searchType){
167:
168:         try{
169:             //setup the searcher variables that are required
170:             setupSearcherAndParser();
171:
172:             //trim off characters that are not useful for searching
173:             queryString = stripNonCharacters(queryString);
174:             queryString = stripStopWords(queryString).toLowerCase();
175:
176:             //the list of hits that we will eventually return
177:             ArrayList<EEKBMatchResult> toReturn = new ArrayList<EEKBMatchResult>();
178:
179:             if(queryString == null || queryString.equals("")){
180:                 return toReturn;
181:             }
182:
183:             //create the query and run the search
184:             Query query = parser.parse(queryString);
185:             TopDocs hits = searcher.search(query, 10);
186:
187:
188:
189:             //move through the returned hits, and convert them back into EEKBNode objects
190:             for ( ScoreDoc scoreDoc : hits.scoreDocs ) {
191:                 Document doc = searcher.doc( scoreDoc.doc );
192:
193:                 //if the returned hit is of the correct type and it is a node
194:                 if(doc.get("type").equals("Node") && searchType.equals("Node")){
195:
196:                     EEKBNode matchingNode = knowledgeBase.getNodeFromId(Integer.parseInt(doc.get("id")));
197:
198:                     //if the hit is not null and not already in our list, then we add it
199:                     if(matchingNode != null && !containsResult(toReturn, matchingNode) && scoreDoc.score > MATCH_SCORE_MIN){
200:
201:                         //create a match result for this node and score
202:                         EEKBMatchResult matchResult = new EEKBMatchResult(matchingNode, scoreDoc.score);
203:
204:                         //add this result to our list that we will return
205:                         toReturn.add(matchResult);
206:                     }
207:                 }
208:                 else if(doc.get("type").equals("Relation") && searchType.equals("Relation")){
209:                     EEKBRelationship matchingRel = knowledgeBase.getRelationById(Integer.parseInt(doc.get("id")));
210:
211:                     //if the hit is not null and not already in our list, then we add it
212:                     if(matchingRel != null && !containsResult(toReturn, matchingRel) && scoreDoc.score > MATCH_SCORE_MIN){
213:
214:                         //create a match result for this node and score
215:                         EEKBMatchResult matchResult = new EEKBMatchResult(matchingRel, scoreDoc.score);
216:
217:                         //add this result to our list that we will return
218:                         toReturn.add(matchResult);
219:                     }
220:                 }
221:             }
222:             searcher.close();
223:
224:             if(humanConfirmMode && queryString.indexOf("supports")==-1 && queryString.indexOf("refutes")==-1){
225:                 //here we ask the user if the match is indeed correct
226:                 //first create the scanner
227:                 Scanner userInput = new Scanner( System.in );
228:
229:                 if(toReturn.size() > 0){
230:
231:                     //show the user the two things that were "matched"
232:                     System.out.println("Searched For: " + queryString);
233:                     System.out.println("Found: " + toReturn.get(0).getResultItem().toString());
```

```
234:
235:                //get the user's input
236:                System.out.println("Is This correct?");
237:
238:                int input = userInput.nextInt();
239:
240:                System.out.println("");
241:
242:                if(input==0)
243:                   numIncorrectMatches++;
244:                else if(input==1)
245:                   numCorrectMatches++;
246:
247:                System.out.println("Stats so far:");
248:                System.out.println("Correct matches: " + numCorrectMatches);
249:                System.out.println("Incorrect matches: " + numIncorrectMatches);
250:                System.out.println("Correct Ignores: " + numCorrectIgnores);
251:                System.out.println("Incorrect Ignores: " + numIncorrectIgnores);
252:                System.out.println("\nTotal: " + (numCorrectMatches + numIncorrectMatches + numCorrectIgnores + numIncorrectIgnores) + "\n");
253:             }
254:             /*else{
255:
256:                System.out.println("No match found for: " + queryString);
257:
258:                //get the user's input
259:                System.out.println("Is This correct?");
260:
261:                int input = userInput.nextInt();
262:
263:                System.out.println("");
264:
265:                if(input==0)
266:                   numIncorrectIgnores++;
267:                else if(input==1)
268:                   numCorrectIgnores++;
269:             }*/
270:
271:
272:          }
273:
274:
275:
276:          //return the list of matches
277:          return toReturn;
278:       }
279:       catch(Exception e){
280:          e.printStackTrace();
281:          return null;
282:       }
283:    }
284:
285:    public void setHumanConfirmMode(boolean value){
286:       humanConfirmMode = value;
287:    }
288:
289:    private String stripNonCharacters(String input){
290:       if (input != null && input.length() > 0){
291:          String propToMatch = input;
292:          propToMatch = propToMatch.trim();
293:          if (propToMatch.length() > 0){
294:             propToMatch = propToMatch.replaceAll("!", "");
295:             propToMatch = propToMatch.replaceAll("\\(", "");
296:             propToMatch = propToMatch.replaceAll("\\)", "");
297:             propToMatch = propToMatch.replaceAll("\\?", "");
298:             propToMatch = propToMatch.replaceAll("\\*", "");
299:             propToMatch = propToMatch.replaceAll(":", "");
300:             propToMatch = propToMatch.replaceAll("\\[", "");
301:             propToMatch = propToMatch.replaceAll("\\]", "");
302:             propToMatch = propToMatch.replaceAll("\"", "");
303:             propToMatch = propToMatch.replaceAll("\'", "");
304:             propToMatch = propToMatch.replaceAll("\\\\", "");
305:             propToMatch = propToMatch.replaceAll("\\{", "");
306:             propToMatch = propToMatch.replaceAll("\\}", "");
307:             propToMatch = propToMatch.replaceAll("\\^", "");
308:             propToMatch = propToMatch.replaceAll("=", "");
309:             propToMatch = propToMatch.replaceAll("-", "");
310:             propToMatch = propToMatch.replaceAll("_", "");
311:          }
```

```
312:
313:            return propToMatch;
314:        }
315:        return "";
316:    }
317:
318:    //function that tests if a node is already in a given list of results (arraylist contains method is not enough)
319:    private boolean containsResult(ArrayList<EEKBMatchResult> matches, EEKBItem newMatch){
320:
321:        //loop through the matches
322:        for(EEKBMatchResult curMatch: matches){
323:            //if the id's are the same then return true
324:            if(curMatch.getResultItem().getId() == newMatch.getId())
325:                return true;
326:        }
327:
328:        return false;
329:    }
330:
331:
332:    private void clearIndexDirectory(){
333:        File directory = new File(INDEX_FILE_PATH);
334:
335:        // Get all files in directory
336:
337:        File[] files = directory.listFiles();
338:        for (File file : files)
339:        {
340:
341:            // Delete each file
342:
343:            if (!file.delete())
344:            {
345:                // Failed to delete file
346:
347:                System.out.println("Failed to delete "+file);
348:            }
349:        }
350:    }
351:
352:    //this method strips stop words and very small words (2 chars or less) from the input
353:    private String stripStopWords(String input){
354:        StringTokenizer tokenizer = new StringTokenizer(input, " ");
355:        String result = "";
356:
357:        while(tokenizer.hasMoreTokens()){
358:            String token = tokenizer.nextToken();
359:            token = token.toLowerCase();
360:            if(!this.stopWords.contains(token) && token.length() > 2){
361:                result += token + " ";
362:            }
363:        }
364:
365:        return result;
366:    }
367:
368:    private static List<String> readStopWordsFromFile(String path){
369:
370:        try {
371:            List<String> stopWords = new ArrayList<String>();
372:            File f = new File(path);
373:
374:            Scanner scanner = new Scanner(new FileReader(f));
375:            while (scanner.hasNext()) {
376:                stopWords.add(scanner.next());
377:            }
378:            return stopWords;
379:        }
380:        catch (Exception e){
381:
382:            return new ArrayList<String>();
383:        }
384:    }
385:
386:    public static List<String> getDefaultStopWords(){
387:        return readStopWordsFromFile("etc/DefaultStopWords.txt");
388:    }
389:
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  import org.jdom.Element;
4:  import org.jdom.Attribute;
5:
6:  import java.util.List;
7:
8:  /**
9:   * Created by IntelliJ IDEA.
10:  * User: mfloryan
11:  * Date: Apr 6, 2012
12:  * Time: 11:13:18 AM
13:  * This class is represents a generic proposition that can be entered into
14:  * out evolving expert knowledge base.
15:  */
16: public class EEKBNode extends EEKBItem{
17:
18:     //INSTANCE PROPERTIES
19:     //------------------------------------------------------------------
20:
21:     //The data associated with this proposition
22:     private String data;
23:
24:     //display data if different than the data above
25:     //sometimes we want to use longer data to display (this var) and shorter data to search on
26:     private String displayData;
27:
28:     //the type of node this represents
29:     private EEKBNodeType nodeType;
30:
31:     //the confidence with which this node is "true"
32:     private double confidence = 10;
33:
34:     private static final int MAX_CONFIDENCE = 100;
35:
36:     public boolean Found = false;
37:
38:
39:     //------------------------------------------------------------------
40:
41:     //CONSTRUCTORS
42:     public EEKBNode(String data, String displayData, EEKBNodeType type, int id){
43:
44:         super(id);
45:         this.displayData = displayData;
46:         this.data = this.stripNonCharacters(data).toLowerCase();
47:         this.nodeType = type;
48:
49:     }
50:
51:     public EEKBNode(String data, EEKBNodeType type, int id){
52:
53:         super(id);
54:         this.data = data;
55:         this.displayData = data;
56:         this.nodeType = type;
57:
58:     }
59:
60:     private EEKBNode(){
61:         super(-1);
62:     }
63:
64:     //increases the confidence of this node
65:     public void increaseConfidence(){
66:         //increase confidence by 3
67:         this.confidence += 3;
68:
69:         //if the confidence is now more then the max, then reset it to the max
70:         if(this.confidence > MAX_CONFIDENCE){
71:             this.confidence = MAX_CONFIDENCE;
72:         }
73:     }
74:
75:     //gets an XML representation of this node
76:     protected Element toXML(){
77:
78:         //make an element for this object
```

```
79:         Element nodeElement = new Element("EEKBNode");
80:
81:         //add the id as an attribute
82:         nodeElement.setAttribute(new Attribute("id", ""+id));
83:
84:         //add the data, nodeType, and confidence as subElements
85:         nodeElement.addContent(new Element("data").setText(data));
86:         nodeElement.addContent(new Element("displayData").setText(displayData));
87:         nodeElement.addContent(new Element("nodeType").setText(nodeType.toString()));
88:         nodeElement.addContent(new Element("confidence").setText(""+confidence));
89:
90:         //return this node element
91:         return nodeElement;
92:     }
93:
94:
95:
96:     protected static EEKBNode fromXML(Element nodeXML){
97:
98:         try{
99:
100:            //create an empty node
101:            EEKBNode newNode = new EEKBNode();
102:
103:            //read the id from the attribute of the nodeXML
104:            newNode.setId(nodeXML.getAttribute("id").getIntValue());
105:
106:            //get the data, nodeType, and confidence of this xml node from the children
107:            newNode.setData(nodeXML.getChildText("data"));
108:            newNode.setDisplayData(nodeXML.getChildText("displayData"));
109:            newNode.setNodeType(EEKBNodeType.stringToNodeType(nodeXML.getChildText("nodeType")));
110:            newNode.setConfidence(Double.parseDouble(nodeXML.getChildText("confidence")));
111:
112:            //return the node
113:            return newNode;
114:        }
115:        catch(Exception e){
116:            e.printStackTrace();
117:            return null;
118:        }
119:    }
120:
121:
122:    /*
123:    *   Makes a deep clone of this node
124:    * */
125:    public EEKBNode clone(){
126:
127:        //new node
128:        EEKBNode newNode = new EEKBNode();
129:
130:        newNode.id = this.id;
131:
132:        //The data associated with this proposition
133:        newNode.data = this.data;
134:
135:        //display data is copied over
136:        newNode.displayData = this.displayData;
137:
138:        //the type of node this represents
139:        newNode.nodeType = this.nodeType;
140:
141:        //the confidence with which this node is "true"
142:        newNode.confidence = this.confidence;
143:
144:        return newNode;
145:
146:    }
147:
148:    public void copy(EEKBNode otherNode){
149:
150:        this.id = otherNode.id;
151:
152:        //The data associated with this proposition
153:        this.data = otherNode.data;
154:
155:        //display data is copied over
156:        this.displayData = otherNode.displayData;
```

```
157:
158:        //the type of node this represents
159:        this.nodeType = otherNode.nodeType;
160:
161:        //the confidence with which this node is "true"
162:        //this.confidence = otherNode.confidence;
163:    }
164:
165:
166:    public String getData() {
167:        return data;
168:    }
169:
170:    public void setData(String data) {
171:        this.data = data;
172:    }
173:
174:    public EEKBNodeType getNodeType() {
175:        return nodeType;
176:    }
177:
178:    public void setNodeType(EEKBNodeType nodeType) {
179:        this.nodeType = nodeType;
180:    }
181:
182:    private void setConfidence(double confidence) {
183:        this.confidence = confidence;
184:    }
185:
186:    public double getConfidence() {
187:        return confidence;
188:    }
189:
190:    public String toString(){
191:
192:        return displayData;
193:
194:    }
195:
196:    public String getDisplayData() {
197:        if(displayData != null)
198:            return displayData;
199:
200:        return data;
201:    }
202:
203:    public void setDisplayData(String displayData) {
204:        this.displayData = displayData;
205:    }
206:
207:    private String stripNonCharacters(String input){
208:        if (input != null && input.length() > 0){
209:            String propToMatch = input;
210:            propToMatch = propToMatch.trim();
211:            if (propToMatch.length() > 0){
212:                propToMatch = propToMatch.replaceAll("!", "");
213:                propToMatch = propToMatch.replaceAll("\\(", "");
214:                propToMatch = propToMatch.replaceAll("\\)", "");
215:                propToMatch = propToMatch.replaceAll("\\?", "");
216:                propToMatch = propToMatch.replaceAll("\\*", "");
217:                propToMatch = propToMatch.replaceAll(":", "");
218:                propToMatch = propToMatch.replaceAll("\\[", "");
219:                propToMatch = propToMatch.replaceAll("\\]", "");
220:                propToMatch = propToMatch.replaceAll("\"", "");
221:                propToMatch = propToMatch.replaceAll("\'", "");
222:                propToMatch = propToMatch.replaceAll("\\\\", "");
223:                propToMatch = propToMatch.replaceAll("\\{", "");
224:                propToMatch = propToMatch.replaceAll("\\}", "");
225:                propToMatch = propToMatch.replaceAll("\\^", "");
226:                propToMatch = propToMatch.replaceAll("=", "");
227:                propToMatch = propToMatch.replaceAll("-", "");
228:                propToMatch = propToMatch.replaceAll("_", "");
229:            }
230:
231:            return propToMatch;
232:        }
233:        return "";
234:    }
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  /**
4:   * Created by IntelliJ IDEA.
5:   * User: mfloryan
6:   * Date: Apr 6, 2012
7:   * Time: 11:42:31 AM
8:   * To change this template use File | Settings | File Templates.
9:   */
10: public enum EEKBNodeType {
11:
12:     hypothesis, data, inference;
13:
14:     public static EEKBNodeType stringToNodeType(String type){
15:
16:         type = type.toLowerCase();
17:
18:         if(type.equals("hypothesis"))
19:             return hypothesis;
20:
21:         if(type.equals("data"))
22:             return data;
23:
24:         if(type.equals("inference"))
25:             return inference;
26:
27:         return null;
28:     }
29: }
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  import org.jdom.Element;
4:  import org.jdom.Attribute;
5:
6:  /**
7:   * Created by IntelliJ IDEA.
8:   * User: mfloryan
9:   * Date: Apr 6, 2012
10:  * Time: 11:50:48 AM
11:  * To change this template use File | Settings | File Templates.
12:  */
13: public class EEKBRelationship extends EEKBItem{
14:
15:    //INSTANCE PROPERTIES
16:    //------------------------------------------------------------
17:
18:
19:    //source and destination nodes
20:    private EEKBNode fromNode;
21:    private EEKBNode toNode;
22:
23:    //relationship type
24:    private EEKBRelationshipType relType;
25:
26:    //confidence that this relationship is "true"
27:    private double confidence = 10;
28:
29:    private static final int MAX_CONFIDENCE = 100;
30:
31:    public boolean Found = false;
32:
33:
34:    public EEKBRelationship(EEKBNode from, EEKBNode to, EEKBRelationshipType relationshipType, int id){
35:
36:       super(id);
37:       fromNode = from;
38:       toNode = to;
39:       relType = relationshipType;
40:
41:    }
42:
43:    private EEKBRelationship(int id){
44:       super(id);
45:
46:    }
47:
48:    //increases the confidence slightly
49:    protected void increaseConfidence(){
50:       this.confidence += 3;
51:
52:       if(confidence > MAX_CONFIDENCE)
53:          confidence = MAX_CONFIDENCE;
54:
55:
56:    }
57:
58:
59:    //gets an XML representation of this node
60:    protected Element toXML(){
61:
62:       //make an element for this object
63:       Element nodeElement = new Element("EEKBRelationship");
64:
65:
66:       //add the data, nodeType, and confidence as subElements
67:       nodeElement.addContent(new Element("id").setText("" + id));
68:       nodeElement.addContent(new Element("fromNodeId").setText("" + fromNode.getId()));
69:       nodeElement.addContent(new Element("toNodeId").setText("" + toNode.getId()));
70:       nodeElement.addContent(new Element("relationshipType").setText(relType.toString()));
71:       nodeElement.addContent(new Element("confidence").setText(""+confidence));
72:
73:       //return this node element
74:       return nodeElement;
75:    }
76:
77:
78:    //convert relationship xml into a relationship object...not sure if this works :(
```

```
79:  //need a reference to the eekb and its nodes in order to do this
80:  protected static EEKBRelationship fromXML(Element relationXML, EvolvingEKB eekbRef){
81:
82:     try{
83:
84:        //create an empty node
85:        EEKBRelationship newRel = new EEKBRelationship(-1);
86:
87:        //pull to and from node id's out of xml and look the nodes up in the eekb
88:        EEKBNode fromNode = eekbRef.getNodeFromId(Integer.parseInt(relationXML.getChildText("fromNodeId")));
89:        EEKBNode toNode = eekbRef.getNodeFromId(Integer.parseInt(relationXML.getChildText("toNodeId")));
90:
91:        //set the id
92:        newRel.setId(Integer.parseInt(relationXML.getChildText("id")));
93:
94:        //set the to and from nodes
95:        newRel.setFromNode(fromNode);
96:        newRel.setToNode(toNode);
97:
98:        //get the relType and confidence
99:        newRel.setRelType(EEKBRelationshipType.getRelTypeFromString(relationXML.getChildText("relationshipType")));
100:       newRel.setConfidence(Double.parseDouble(relationXML.getChildText("confidence")));
101:
102:       //return the node
103:       return newRel;
104:    }
105:    catch(Exception e){
106:       e.printStackTrace();
107:       return null;
108:    }
109: }
110:
111:
112: /*
113: * Makes a deep clone of this relationship
114: * */
115: public EEKBRelationship clone(){
116:
117:    //make a new rel
118:    EEKBRelationship newRel = new EEKBRelationship(this.getId());
119:
120:    newRel.fromNode = this.fromNode.clone();
121:    newRel.toNode = this.toNode.clone();
122:
123:    //relationship type
124:    newRel.relType = this.relType;
125:
126:    //confidence that this relationship is "true"
127:    newRel.confidence = this.confidence;
128:
129:    return newRel;
130: }
131:
132: public void copy(EEKBRelationship otherRel){
133:
134:    this.fromNode.copy(otherRel.fromNode);
135:    this.toNode.copy(otherRel.toNode);
136:
137:    this.relType = otherRel.relType;
138:
139:    //this.confidence = otherRel.confidence;
140:
141: }
142:
143:
144: //-------------------------------------------------------------
145:
146: public EEKBNode getFromNode() {
147:    return fromNode;
148: }
149:
150: public void setFromNode(EEKBNode fromNode) {
151:    this.fromNode = fromNode;
152: }
153:
154: public EEKBNode getToNode() {
155:    return toNode;
156: }
```

```
157:
158:    public void setToNode(EEKBNode toNode) {
159:       this.toNode = toNode;
160:    }
161:
162:    public EEKBRelationshipType getRelType() {
163:       return relType;
164:    }
165:
166:    public void setRelType(EEKBRelationshipType relType) {
167:       this.relType = relType;
168:    }
169:
170:    public double getConfidence() {
171:       return confidence;
172:    }
173:
174:    public void setConfidence(double confidence) {
175:       this.confidence = confidence;
176:    }
177:
178:    public String toString(){
179:
180:       return fromNode.getDisplayData() + " " + relType + " " + toNode.getDisplayData();
181:
182:    }
183: }
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  /**
4:   * Created by IntelliJ IDEA.
5:   * User: mfloryan
6:   * Date: Apr 6, 2012
7:   * Time: 11:49:51 AM
8:   * To change this template use File | Settings | File Templates.
9:   */
10: public enum EEKBRelationshipType {
11:     supports, refutes, weaklyRefutes, stronglySupports, isConsistentWith, nuetral, unknown, almostTheSame, none;
12:
13:
14:
15:
16:     //this function takes in a relationship type as a string and converts it to the enumeration
17:     //version of itself.
18:
19:     //Prints out a message if the given type is not one of the supported types
20:     public static EEKBRelationshipType getRelTypeFromString(String relType){
21:
22:         relType = relType.toLowerCase();
23:
24:         if(relType.equals("supports")){
25:             return EEKBRelationshipType.supports;
26:         }
27:         else if(relType.equals("refutes")){
28:             return EEKBRelationshipType.refutes;
29:         }
30:         else if(relType.equals("isconsistentwith") || relType.equals("is consistent with")){
31:             return EEKBRelationshipType.isConsistentWith;
32:         }
33:         else if(relType.equals("neutral") || relType.equals("nuetral")){
34:             return EEKBRelationshipType.nuetral;
35:         }
36:         else if(relType.equals("weaklyrefutes") || relType.equals("weakly refutes")){
37:             return EEKBRelationshipType.weaklyRefutes;
38:         }
39:         else if(relType.equals("strongly supports") || relType.equals("stronglysupports")){
40:             return EEKBRelationshipType.stronglySupports;
41:         }
42:         else if(relType.equals("unknown")){
43:             return EEKBRelationshipType.unknown;
44:         }
45:         else if(relType.equals("is almost the same as")){
46:             return EEKBRelationshipType.almostTheSame;
47:         }
48:         else if(relType.equals("none")){
49:             return EEKBRelationshipType.none;
50:         }
51:         else{
52:             System.out.println("WARNING in EEKBTester.getRelTypeFromString: trying to convert a relationship type that does not exist: " + relTy
pe + ". Returning 'supports' by default");
53:             return EEKBRelationshipType.supports;
54:         }
55:
56:     }
57: }
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  import edu.umass.ckc.rashi.ekb.ExpertKnowledgeBase;
4:  import edu.umass.ckc.rashi.Matching.Matcher;
5:
6:  import java.util.ArrayList;
7:  import java.io.BufferedWriter;
8:  import java.io.FileWriter;
9:
10: import org.jdom.Element;
11: import org.jdom.Attribute;
12: import org.jdom.Document;
13: import org.jdom.output.XMLOutputter;
14:
15: /**
16:  * Created by IntelliJ IDEA.
17:  * User: mfloryan
18:  * Date: Apr 6, 2012
19:  * Time: 10:44:51 AM
20:  * To change this template use File | Settings | File Templates.
21:  */
22: public class EvolvingEKB {
23:
24:     //INSTANCE PROPERTIES
25:     //------------------------------------------------------------------------
26:
27:     //A set of nodes for this Evolving EKB
28:     private ArrayList<EEKBNode> nodes;
29:
30:     //a set of relationships between these nodes
31:     private ArrayList<EEKBRelationship> relations;
32:
33:     //matcher used for determining if a new addition to this
34:     //knowledge base is already present
35:     private EEKBMatcher matcher;
36:
37:     //This object is used to create unique Identification Numbers for new elements
38:     private IDTracker idTracker;
39:
40:     //the output filename and possible extensions for saving this EEKB
41:     private static final String SAVE_FILE_NAME = "dataFiles/EEKB_Output/EvolvingEKB";
42:     private static final String SAVE_DATA_EXTENSION = ".dat";
43:     private static final String SAVE_XML_EXTENSION = ".xml";
44:
45:
46:     //the minimum confidence that must be reached for an element to be accepted in the final EEKB
47:     private static final int CONFIDENCE_THRESHOLD = 15;
48:
49:     //counts the total pieces of evidence considered
50:     private int evidenceCount = 0;
51:
52:
53:
54:
55:
56:     //The target domain (Arguably not necessary here may add later)!
57:
58:     //------------------------------------------------------------------------
59:
60:     //constructor
61:     public EvolvingEKB(){
62:         nodes = new ArrayList<EEKBNode>();
63:         relations = new ArrayList<EEKBRelationship>();
64:
65:         //initialize the ID number tracker
66:         idTracker = new IDTracker();
67:
68:         //initialize the matcher
69:         matcher = new EEKBMatcher(this);
70:     }
71:
72:
73:     //version that does not accept the display data variable
74:     public int nodeEvidence(String nodeData, EEKBNodeType type){
75:
76:         return this.nodeEvidence(nodeData, nodeData, type);
77:     }
78:
```

```
79:
80:
81:     //This function attempts to add a new node to this EEKB
82:     //determines whether the node
83:     //this function should return the internal ID number of the node that is created
84:     public int nodeEvidence(String nodeData, String displayData, EEKBNodeType type){
85:
86:         this.evidenceCount++;
87:
88:
89:         //if we believe this data is already in the EEKB
90:         if(this.alreadyContainsData(nodeData)){
91:
92:             //get the node that is already present
93:             EEKBNode presentNode = this.getTopNodeSearchResult(nodeData);
94:
95:             //if the new node has a more succinct description, then use that instead
96:             /*if(nodeData.length() >=3 && nodeData.length() < presentNode.getData().length()){
97:
98:                 //overwrite the data and display data with the new stuff
99:                 presentNode.setData(nodeData);
100:                presentNode.setDisplayData(displayData);
101:            } */
102:
103:            //System.out.println("Found a match! Combining: " + nodeData + " with " + presentNode.getData());
104:
105:            //Only combine these nodes if their types are the same
106:            if(presentNode.getNodeType() == type){
107:
108:                //increase this nodes confidence
109:                presentNode.increaseConfidence();
110:
111:                //rebuild the search index
112:                matcher.rebuildIndex(this);
113:
114:                return presentNode.getId();
115:            }
116:        }
117:
118:        //get a new id num
119:        int idNum = idTracker.getNextAvailableIDNumber();
120:
121:        //setup the node
122:        EEKBNode newNode = new EEKBNode(nodeData, displayData, type, idNum);
123:
124:        //add this node to the list of nodes
125:        this.nodes.add(newNode);
126:
127:        //rebuild the search index
128:        matcher.rebuildIndex(this);
129:
130:        return idNum;
131:
132:    }
133:
134:
135:    public void relationshipEvidence(int from, int to, EEKBRelationshipType relType){
136:        //for now, let's just print out the evidence that comes in
137:        //System.out.print("NEW RELATIONSHIP:  ");
138:        //System.out.println(relType + ": " + from + " --> " + to);
139:        this.evidenceCount++;
140:
141:        if(!this.alreadyContainsRelationship(getNodeFromId(from), getNodeFromId(to), relType)){
142:
143:            //get a new id num
144:            int idNum = idTracker.getNextAvailableIDNumber();
145:
146:            //create a new relationship node
147:            EEKBRelationship newRelation = new EEKBRelationship(getNodeFromId(from), getNodeFromId(to), relType, idNum);
148:
149:            this.relations.add(newRelation);
150:        }
151:        else{
152:
153:            //get the matching relationship
154:            EEKBRelationship rel = this.getTopRelSearchResult(getNodeFromId(from), getNodeFromId(to), relType);
155:
156:            //increase its confidence
```

```
157:            rel.increaseConfidence();
158:        }
159:
160:        matcher.rebuildIndex(this);
161:    }
162:
163:
164:    //This function searches our set of nodes and attempts to find the one that
165:    //has the given nodeId
166:    public EEKBNode getNodeFromId(int nodeId){
167:
168:        //for every node in our EEKB
169:        for(EEKBNode node: nodes){
170:
171:            //if the id of this node is the one we are searching for, then return it
172:            if(node.getId() == nodeId){
173:                return node;
174:            }
175:        }
176:
177:        System.out.println("WARNING in EvolvingEKB.getNodeFromId:  Cannot find node with id " + nodeId + ". Returning 'null' by default");
178:        return null;
179:    }
180:
181:    public EEKBRelationship getRelationById(int relationId){
182:
183:        //for every relation in our EEKB
184:        for(EEKBRelationship rel: relations){
185:
186:            //if the id of this node is the one we are searching for, then return it
187:            if(rel.getId() == relationId){
188:                return rel;
189:            }
190:        }
191:
192:        System.out.println("WARNING in EvolvingEKB.getRelationById:  Cannot find relation with id " + relationId + ". Returning 'null' by default
");
193:        return null;
194:
195:    }
196:
197:    //This function takes in an EEKB Node and returns all of the relationships that are
198:    //associated with that node
199:    private ArrayList<EEKBRelationship> getAllRelationsForNode(EEKBNode node){
200:        //The array we will be returning
201:        ArrayList<EEKBRelationship> toReturn = new ArrayList<EEKBRelationship>();
202:
203:        //for every relation that exists in our EEKB
204:        for(EEKBRelationship rel: relations){
205:            //if either the from or to node id is equal to the parameter node's id, then add it to the list to return
206:            if(rel.getFromNode().getId() == node.getId() || rel.getToNode().getId() == node.getId()){
207:                toReturn.add(rel);
208:            }
209:        }
210:
211:        //return the list of found relationships
212:        return toReturn;
213:    }
214:
215:    //This function takes in an EEKB Node and returns all of the relationships that are
216:    //directed TO the node
217:    private ArrayList<EEKBRelationship> getRelationsToNode(EEKBNode node){
218:        //The array we will be returning
219:        ArrayList<EEKBRelationship> toReturn = new ArrayList<EEKBRelationship>();
220:
221:        //for every relation that exists in our EEKB
222:        for(EEKBRelationship rel: relations){
223:            //if to node id is equal to the parameter node's id, then add it to the list to return
224:            if(rel.getToNode().getId() == node.getId()){
225:                toReturn.add(rel);
226:            }
227:        }
228:
229:        //return the list of found relationships
230:        return toReturn;
231:    }
232:
233:    //This function takes in a node, and returns any node that we believe is the same as the one given
```

```
234:    //use matcher tells us whether to use the smart matcher or just to use string equality only
235:    protected ArrayList<EEKBNode> searchForNode(String nodeData, boolean useMatcher){
236:
237:        nodeData = this.stripNonCharacters(nodeData).toLowerCase();
238:
239:        //the list that will be returned
240:        ArrayList<EEKBNode> toReturn = new ArrayList<EEKBNode>();
241:
242:        //search through our current set of nodes
243:        for(EEKBNode node: nodes){
244:
245:            //if node data is the same as the parameter, then return the found node
246:            if(node.getData().toLowerCase().equals(nodeData)){
247:                toReturn.add(node);
248:                return toReturn;
249:            }
250:        }
251:
252:
253:        //if the user wants to use the matcher, then we attempt to do a smarter search
254:        if(useMatcher){
255:            //use our matcher to find potential nodes within the EEKB that match this new piece of data
256:            ArrayList<EEKBMatchResult> matches = matcher.performSearch(nodeData, "Node");
257:
258:            //if there is more than one match, then we return something
259:            if(matches != null && matches.size() > 0){
260:
261:                //add every match to the list and return the list
262:                for(EEKBMatchResult match : matches){
263:
264:                    toReturn.add((EEKBNode)match.getResultItem());
265:                }
266:
267:                return toReturn;
268:            }
269:        }
270:
271:        return null;
272:    }
273:
274:    //overwritten method, uses the matcher by default
275:    protected ArrayList<EEKBNode> searchForNode(String nodeData){
276:        return this.searchForNode(nodeData, true);
277:    }
278:
279:    protected EEKBNode getTopNodeSearchResult(String nodeData){
280:
281:        ArrayList<EEKBNode> matches = this.searchForNode(nodeData,true);
282:
283:        if(matches != null && matches.size()>0)
284:            return matches.get(0);
285:
286:        return null;
287:    }
288:
289:    protected EEKBRelationship getTopRelSearchResult(EEKBNode from, EEKBNode to, EEKBRelationshipType type){
290:
291:        ArrayList<EEKBRelationship> matches = this.searchForRelationship(from, to, type);
292:
293:        if(matches != null && matches.size()>0)
294:            return matches.get(0);
295:
296:        return null;
297:    }
298:
299:    //Returns true if we think that nodeData is already represented in our EEKB
300:    protected boolean alreadyContainsData(String nodeData){
301:
302:        //we return true iff our search function returns something other than null
303:        return searchForNode(nodeData) != null;
304:    }
305:
306:    //Returns true if we think that nodeData is already represented in our EEKB
307:    protected boolean alreadyContainsData(String nodeData, boolean useMatcher){
308:
309:        //we return true iff our search function returns something other than null
310:        return searchForNode(nodeData, useMatcher) != null;
311:    }
```

```
312:
313:
314:    //Function searches for a given relationship in our EEKB and returns if we think it already exists
315:    protected ArrayList<EEKBRelationship> searchForRelationship(EEKBNode fromNode, EEKBNode toNode, EEKBRelationshipType relType, boolea
n useMatcher){
316:
317:        ArrayList<EEKBRelationship> toReturn = new ArrayList<EEKBRelationship>();
318:
319:        //the simple part, loop through our relationships and see if this one exists already!
320:        for(EEKBRelationship relation: relations){
321:
322:            //if the fromId, toId, and type are all the same, then return it!
323:            if(fromNode.getId() == relation.getFromNode().getId() && toNode.getId() == relation.getToNode().getId()){
324:                //System.out.println("Found relationship match!");
325:                if(! (relType == relation.getRelType())){
326:                    //System.out.println("But the types are different!");
327:                }
328:
329:                toReturn.add(relation);
330:                return toReturn;
331:            }
332:        }
333:
334:
335:        //if it is not found the easy way, then we try to use the matcher
336:        if(useMatcher){
337:
338:            //a dummy relation
339:            EEKBRelationship dummy = new EEKBRelationship(fromNode,toNode,relType,-1);
340:
341:            //System.out.println("Performing search for: " + dummy.toString());
342:
343:            //use our matcher to find potential nodes within the EEKB that match this new piece of data
344:            ArrayList<EEKBMatchResult> matches = matcher.performSearch(dummy.toString(), "Relation");
345:
346:            //System.out.println("Searching for relationship for: " + dummy.toString());
347:            //if there is more than one match, then we return something
348:            if(matches != null && matches.size() > 0){
349:
350:
351:                for(EEKBMatchResult match : matches){
352:
353:                    toReturn.add((EEKBRelationship)match.getResultItem());
354:                }
355:
356:                //return the top match
357:                return toReturn;
358:            }
359:        }
360:
361:        return null;
362:    }
363:
364:    protected ArrayList<EEKBRelationship> searchForRelationship(EEKBNode fromNode, EEKBNode toNode, EEKBRelationshipType relType){
365:        return this.searchForRelationship(fromNode, toNode, relType, true);
366:    }
367:
368:    //This function simply searches for the relationship using the string representation
369:    //of that relationship
370:    protected EEKBRelationship searchForRelationship(String relString){
371:
372:        //pass it into the matcher
373:        ArrayList<EEKBMatchResult> results = matcher.performSearch(relString, "Relation");
374:
375:        //see if there are any results
376:        if(results.size() > 0){
377:
378:            //return the first one
379:            return (EEKBRelationship)results.get(0).getResultItem();
380:        }
381:
382:        //return null if the matches are empty
383:        return null;
384:    }
385:
386:
387:    //function returns true iff we believe that the given relationship already exists.
388:    protected boolean alreadyContainsRelationship(EEKBNode fromNode, EEKBNode toNode, EEKBRelationshipType relType, boolean useMatcher){
```

```
389:
390:        //return true iff our search returns something
391:        return searchForRelationship(fromNode, toNode, relType, useMatcher) != null;
392:    }
393:
394:    //function returns true iff we believe that the given relationship already exists.
395:    protected boolean alreadyContainsRelationship(EEKBNode fromNode, EEKBNode toNode, EEKBRelationshipType relType){
396:
397:        //return true iff our search returns something
398:        return searchForRelationship(fromNode, toNode, relType) != null;
399:    }
400:
401:
402:    //Prints out the EEKB!
403:    public void print(){
404:
405:        System.out.println("-------------------------------------------------");
406:        System.out.println("PRINTING EEKB");
407:        System.out.println("Total Nodes: " + nodes.size());
408:        System.out.println("Total Relationships: " + relations.size());
409:        System.out.println("-------------------------------------------------");
410:        System.out.println("-------------------------------------------------\n");
411:
412:        //for every node in our EEKB
413:        for(EEKBNode node: nodes){
414:
415:            //print out the node
416:            System.out.println(node.toString());
417:
418:            //get the relationships associated with this node
419:            ArrayList<EEKBRelationship> rels = getRelationsToNode(node);
420:
421:            //if there is at least one relationship going TO this node, then print it out
422:            if(rels.size() > 0){
423:
424:                //print out all of relationships going TO node
425:                for(EEKBRelationship rel: rels){
426:                    System.out.println("" + rel.toString());
427:                }
428:
429:                System.out.println("\n");
430:            }
431:        }
432:
433:        System.out.println("-------------------------------------------------");
434:        System.out.println("-------------------------------------------------");
435:    }
436:
437:    public void printNodes(){
438:
439:        System.out.println("-------------------------------------------------");
440:        System.out.println("PRINTING EEKB Nodes");
441:        System.out.println("Total Nodes: " + nodes.size());
442:        System.out.println("-------------------------------------------------");
443:        System.out.println("-------------------------------------------------\n");
444:
445:        //for every node in our EEKB
446:        for(EEKBNode node: nodes){
447:
448:            if(!node.Found)
449:                //print out the node
450:                System.out.println(node.getId() + ": " + node.toString());
451:
452:        }
453:
454:    }
455:
456:    public void printRelations(){
457:
458:        System.out.println("-------------------------------------------------");
459:        System.out.println("PRINTING EEKB Relations");
460:        System.out.println("Total Relations: " + relations.size());
461:        System.out.println("-------------------------------------------------");
462:        System.out.println("-------------------------------------------------\n");
463:
464:        //for every node in our EEKB
465:        for(EEKBRelationship rel: relations){
466:
```

```
467:            if(!rel.Found)
468:               //print out the node
469:               System.out.println(rel.getId() + ": " + rel.toString());
470:
471:
472:         }
473:
474:      }
475:
476:      /*                    s
477:      * Returns the size of this EEKB
478:      * */
479:      public int getSize(){
480:
481:         //return number of nodes + number of relations
482:         return nodes.size() + relations.size();
483:      }
484:
485:      //Saves the EEKB to an external file
486:      public void saveEEKB(){
487:
488:         try{
489:            // Create file
490:            FileWriter fstream = new FileWriter(SAVE_FILE_NAME + SAVE_XML_EXTENSION);
491:            BufferedWriter out = new BufferedWriter(fstream);
492:
493:            XMLOutputter serializer = new XMLOutputter();
494:            serializer.output(this.toXML(), out);
495:
496:
497:            //Close the output stream
498:            out.close();
499:
500:         }catch (Exception e){//Catch exception if any
501:            e.printStackTrace();
502:         }
503:      }
504:
505:      /*
506:      * Converts this knowledge base to xml
507:      * */
508:      private Document toXML(){
509:         //make an element for this knowledge base
510:         Element eekbElement = new Element("EvolvingEKB");
511:
512:         //for every node, add a new xml element for that node
513:         for(EEKBNode node: nodes){
514:            eekbElement.addContent(node.toXML());
515:         }
516:
517:         //do the same for every relationship
518:         for(EEKBRelationship relation: relations){
519:            eekbElement.addContent(relation.toXML());
520:         }
521:
522:         //returnt the xml
523:         return new Document(eekbElement);
524:      }
525:
526:      public static EvolvingEKB fromXML(Document xml){
527:
528:         if(xml == null){
529:            System.out.println("xml Document is null!");
530:         }
531:
532:
533:         //create the eekb that will be returned...eventually
534:         EvolvingEKB eekb = new EvolvingEKB();
535:
536:         //get the root element and loop through its children
537:         Element root = xml.getRootElement();
538:
539:         System.out.println("Processing Nodes");
540:
541:         //for each EEKBNode child
542:         for(Object nodeChild: root.getChildren("EEKBNode")){
543:
544:            //construct the node from the xml
```

```
545:            EEKBNode newNode = EEKBNode.fromXML((Element)nodeChild);
546:
547:            //add it to the eekb
548:            eekb.nodes.add(newNode);
549:        }
550:
551:        System.out.println("Processing Relations");
552:
553:        //now we do the same for relationships
554:        for(Object relChild: root.getChildren("EEKBRelationship")){
555:
556:            //construct the rel
557:            EEKBRelationship rel = EEKBRelationship.fromXML((Element) relChild, eekb);
558:
559:
560:            //add it if the relation type is not "none"
561:            if(rel.getRelType() != EEKBRelationshipType.none && rel.getRelType() != EEKBRelationshipType.nuetral)
562:                eekb.relations.add(rel);
563:        }
564:
565:        System.out.println("done");
566:        System.out.println("num nodes: " + eekb.getNodes().size());
567:        System.out.println("rel nodes: " + eekb.getRelations().size());
568:
569:        //return the eekb
570:        return eekb;
571:    }
572:
573:
574:    /*
575:    *  This function returns a copy of this EEKB with only high confidence nodes included
576:    *  the confidence threshold is defined by the private final int CONFIDENCE_THRESHOLD
577:    */
578:    public EvolvingEKB getAcceptedKnowledgeBase(){
579:
580:        //first create a new EEKB
581:        EvolvingEKB highConfKnowledge = new EvolvingEKB();
582:
583:        //go through the nodes of this EEKB and add high confidence ones to the new EEKB
584:        for(EEKBNode node: nodes){
585:
586:            //if the confidence is high enough
587:            if(node.getConfidence() >= CONFIDENCE_THRESHOLD){
588:                //add the node to the new EEKB
589:                highConfKnowledge.nodes.add(node.clone());
590:            }
591:        }
592:
593:
594:        //now we go through every relationship
595:        for(EEKBRelationship relation: relations){
596:
597:            //if this relations and its corresponding nodes have high enough confidence
598:            //then we include it
599:            if(relation.getToNode().getConfidence()>=CONFIDENCE_THRESHOLD && relation.getFromNode().getConfidence()>=CONFIDENCE_THRE
SHOLD){
600:            //if(relation.getConfidence() >= CONFIDENCE_THRESHOLD) {
601:                //add it
602:                highConfKnowledge.relations.add(relation.clone());
603:            }
604:        }
605:
606:        highConfKnowledge.rebuildSearchIndex();
607:
608:        return highConfKnowledge;
609:    }
610:
611:    private void rebuildSearchIndex(){
612:
613:        matcher.rebuildIndex(this);
614:    }
615:
616:
617:    /*
618:    * This function reverts the data from this EEKB to match an older one.
619:    * used for the KRG analysis that required human data content changes to be reverted
620:    * */
621:    public void revertContentChanges(EvolvingEKB originalEEKB){
```

```
622:
623:
624:        //loop through the nodes
625:        for(EEKBNode node:nodes){
626:
627:            //if a node with this id is in the original
628:            EEKBNode originalNode = originalEEKB.getNodeFromId(node.getId());
629:
630:            if(node.getId()==28)
631:                System.out.println("Stop");
632:
633:            if(originalNode!=null){
634:
635:                node.copy(originalNode);
636:            }
637:
638:        }
639:
640:        //now for the relations
641:        for(EEKBRelationship rel:relations){
642:
643:            EEKBRelationship originalRel = originalEEKB.getRelationById(rel.getId());
644:
645:            if(originalRel!=null){
646:
647:                rel.copy(originalRel);
648:            }
649:        }
650:    }
651:
652:
653:
654:    //GETTERS AND SETTERS
655:
656:    protected ArrayList<EEKBNode> getNodes() {
657:        return nodes;
658:    }
659:
660:    protected ArrayList<EEKBRelationship> getRelations() {
661:        return relations;
662:    }
663:
664:    public int getEvidenceCount(){
665:        return this.evidenceCount;
666:    }
667:
668:    public void setHumanConfirmMode(boolean value){
669:        matcher.setHumanConfirmMode(value);
670:    }
671:
672:    private String stripNonCharacters(String input){
673:        if (input != null && input.length() > 0){
674:            String propToMatch = input;
675:            propToMatch = propToMatch.trim();
676:            if (propToMatch.length() > 0){
677:                propToMatch = propToMatch.replaceAll("!", "");
678:                propToMatch = propToMatch.replaceAll("\\(", "");
679:                propToMatch = propToMatch.replaceAll("\\)", "");
680:                propToMatch = propToMatch.replaceAll("\\?", "");
681:                propToMatch = propToMatch.replaceAll("\\*", "");
682:                propToMatch = propToMatch.replaceAll(":", "");
683:                propToMatch = propToMatch.replaceAll("\\[", "");
684:                propToMatch = propToMatch.replaceAll("\\]", "");
685:                propToMatch = propToMatch.replaceAll("\"", "");
686:                propToMatch = propToMatch.replaceAll("\'", "");
687:                propToMatch = propToMatch.replaceAll("\\\\", "");
688:                propToMatch = propToMatch.replaceAll("\\{", "");
689:                propToMatch = propToMatch.replaceAll("\\}", "");
690:                propToMatch = propToMatch.replaceAll("\\^", "");
691:                propToMatch = propToMatch.replaceAll("=", "");
692:                propToMatch = propToMatch.replaceAll("-", "");
693:                propToMatch = propToMatch.replaceAll("_", "");
694:            }
695:
696:            return propToMatch;
697:        }
698:        return "";
699:    }
```

```
 1: package edu.umass.ckc.rashi.eekb;
 2:
 3: /**
 4:  * Created by IntelliJ IDEA.
 5:  * User: mfloryan
 6:  * Date: May 19, 2012
 7:  * Time: 10:45:13 AM
 8:  * This class tracks the internal IDs of nodes in the EEKB, can return
 9:  * the next available ID whenever necessary
10:  */
11: public class IDTracker {
12:
13:     //The next available identification number for EEKB elements
14:     private int nextAvailableId = 1;
15:
16:
17:     //empty constructor
18:     public IDTracker(){
19:
20:     }
21:
22:     //Returns the next available ID number for EEKB Elements
23:     public int getNextAvailableIDNumber(){
24:         int toReturn = nextAvailableId;
25:         nextAvailableId++;
26:
27:         return toReturn;
28:     }
29: }
```

```
1:  package edu.umass.ckc.rashi.eekb;
2:
3:  import org.apache.log4j.Logger;
4:  import org.apache.lucene.analysis.*;
5:  import org.apache.lucene.analysis.standard.StandardFilter;
6:  import org.apache.lucene.analysis.standard.StandardTokenizer;
7:  import org.apache.lucene.util.Version;
8:
9:  import java.io.File;
10: import java.io.FileReader;
11: import java.io.Reader;
12: import java.util.ArrayList;
13: import java.util.HashSet;
14: import java.util.List;
15: import java.util.Scanner;
16:
17: /**
18:  * User: dragon
19:  * Date: Jan 25, 2010
20:  * Time: 2:34:28 PM
21:  */
22: public class LuceneStopWordAnalyzer extends Analyzer {
23:     static Logger logger = Logger.getLogger(LuceneStopWordAnalyzer.class);
24:
25:     private HashSet stopTable;
26:     private List<String> stopList;
27:     //private ConflationMapping conflationMapping = new ConflationMapping()
28:
29:     public LuceneStopWordAnalyzer() {
30:         super();
31:
32:         stopList = new ArrayList<String>();
33:
34:         stopTable = new HashSet(StopFilter.makeStopSet(stopList));
35:     }
36:
37:
38:
39:     public TokenStream tokenStream(String s, Reader reader) {
40:         TokenStream result = new StandardTokenizer(Version.LUCENE_30, reader);
41:
42:         result = new StandardFilter(result);
43:         result = new LowerCaseFilter(result);
44:
45:         //false means we are not "recording removed stop words" per Lucene 3.0 api
46:         result = new StopFilter(false, result, stopTable);
47:
48: //        result = new ConflationMapping(caseResourcePath, result);
49:         result = new PorterStemFilter(result);
50:         return result;
51:     }
52:
53:     private static List<String> readStopWordsFromFile(String path){
54:
55:         try {
56:             List<String> stopWords = new ArrayList<String>();
57:             File f = new File(path);
58:
59:             Scanner scanner = new Scanner(new FileReader(f));
60:             while (scanner.hasNext()) {
61:                 stopWords.add(scanner.next());
62:             }
63:             return stopWords;
64:         }
65:         catch (Exception e){
66:             logger.error("In LuceneStopWordAnalyzer.readStopWordsFromFile: Error loading from path - " + path);
67:             return new ArrayList<String>();
68:         }
69:     }
70:
71:     public static List<String> getDefaultStopWords(){
72:         return readStopWordsFromFile("etc/DefaultStopWords.txt");
73:     }
74: }
```

BIBLIOGRAPHY

[1] Aleven, V. *"Using background knowledge in case-based legal reasoning: a computational model and an intelligent learning environment."* Artificial Intelligence 150, 183–237 (2003)

[2] Aleven, V., Ashley, K., Lynch, C., Pinkwart, N.: Proc. ITS for Ill-Defined Domains Work- shop. ITS 2006 (2006)

[3] Aleven, V., Ashley, K., Lynch, C., Pinkwart, N.: Proc. Workshop on AIED applications in ill-defined domains. AIED 2007, Los Angeles, USA (2007)

[4] Aleven, V., Ashley, K., Lynch, C., Pinkwart, N.: Proc. ITS for Ill-Defined Domains Work- shop. ITS 2008, Montreal, Canada (2008)

[5] Amazon's Mechanical Turk. Copyright Amazon.com inc. or Affiliates (2005 – 1012). https://www.mturk.com/mturk/welcome

[6] Arroyo, Ivon; Royer, James M. Woolf, Beverly Park . *"Using an Intelligent Tutor and Math Fluency Training to Improve Math Performance."* International Journal of Artificial Intelligence in Education (2011).

[7] Arroyo, Ivon; Beverly P. Woolf, David Cooper, Winslow Burleson, Kasia Muldner *"The Impact of Animated Pedagogical Agents on Girls' and Boys' Emotions, Attitudes, Behaviors and Learning." International Conference of Advanced Learning Technologies (ICALT 2011) Athens, Georgia, July 2011*

[8] Ashley, K.D., Desai, R., Levine, J.M.: Teaching Case-Based Argumentation Concepts Us- ing Dialectic Arguments vs. Didactic Explanations. In: Cerri, S.A., Gouardéres, G., Paraguaçu, F. (eds.) ITS 2002. LNCS, vol. 2363, pp. 585–595. Springer, Heidelberg (2002)

[9] Baader, Franz; Sertkaya, Baris; *"Usability Issues in Description Logic Knowledge Base Completion"*. Formal Concept Analysis. Lecture Notes in Computer Science, 2009. Volume 5548/2009.

[10] Bader-Natal, Ari. *"Incorporating Game Mechanics into a Network of Online Study Groups"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[11] Beaubien JM, Baker DP. *"The use of simulation for training teamwork skills in health care: how low can you go?"*. Qual Saf Health Care. 2004;13(Suppl 1):i51–56.

[12] Becker, Katrin *"Design Paradox: Instructional Games"*. The International Conference on the Future of Game Design and Technology, The University of Western Ontario. London, Ontario, Canada, October 10 - 12, 2006.

[13] Becker, Katrin. *"Classifying Learning Objectives in Commercial Video Games"*. LDG; Volume 1, No. 1 (2007).

[14] Blessing, S.B. (2003). A programming by demonstration authoring tool for model-tracing tutors. In T. Murray, S. Blessing, & S. Ainsworth (Eds.) *Authoring Tools for Advanced Learning Environments* (pp. 93-119). Dordrecht, the Netherlands: Kluwer Academic Publishers.

[15] Blessing, S., Gilbert, S., Ourada, S., & Ritter, S. (2007). Lowering the bar for creating model-tracing intelligent tutoring systems. In R. Luckin, K. Koedinger, & J. Greer (Eds.), *Proceedings of the 13th International Conference on Artificial Intelligence in Education* (pp. 443–450). Amsterdam, the Netherlands: IOS Press.

[16] de Castell, Suzanne; Jenson, Jennifer; Taylor, Nicholas . *"Digital Games for Education: When Meanings Play"*. In Proceedings of the 2007 Digital Games Research Association Conference. Tokyo, Japan.

[17] Chamberlain, Jon; AnaWiki; *"Phrase Detectives"* (2008 - 2010). http://anawiki.essex.ac.uk/phrasedetectives/

[18] Chou, C.-Y., Chan, T.-W., Lin, C.-J.: Redefining the learning companion: The past, pre- sent, and future of educational agents. Comput. and Educ. 40, 255–269 (2003)

[19] Clancey, W.: Use of MYCIN's rules for tutoring. In: Buchanan, B., Shortliffe, E.H. (eds.) Rule-Based Expert Systems. Addison-Wesley, Reading (1984)

[20] Colby, Rebekah Shultz; Colby, Richard. *"A Pedagogy of Play: Integrating Computer Games into the Writing Classroom"*. Computers and Composition; Volume 25, Issue 3 (2008): Pages 300-312.

[21] Cooper, S., Khatib, F., Treuille, A., Barbero, J., Lee, J., Beenen, M., Leaver-Fay, A., Baker, D., & Popovic, Z. Predicting protein structures with a multiplayer online game. *Nature*, 466:7307, (Aug. 2010), 756–760.

[22] "*Crowdsourcing*". From Wikipedia. http://en.wikipedia.org/wiki/Crowdsourcing. Accessed on Sep. 28 2011.

[23] Culp, Katie McMillan. *"Response to Learning Context: Gaming Simulations, and Science Learning in the Classroom"*. In Committee for Learning Science: Computer Games, Simulations, and Education 2-day workshop October 6-7, 2009. Washington, D.C.

[24] Csikszentmihalyi, M. (1997). Flow and education. *NAMTAjoumal,* 22(2), 2-35

[25] Csikszentmihalyi, Mihalyi (1977), Beyond Boredom and Anxiety, secondprinting. SanFrancisco:Jossey-Bass.

[26] Csikszentlmihalyi, Mihalyi (1990), Flow: The Psychology of Optimal Experience, New York: Harper and Row.

[27] Csikszentmihalyi, Mihalyi and Judith LeFevre (1989), "Optimal Experience in Work and Leisure," Journal of Personality and Social Psychology, 56 (5), 815-822.

[28] Dede, Chris. *"Learning Context: Gaming, Simulations, and Science Learning in the Classroom".* In Committee for Learning Science: Computer Games, Simulations, and Education 2-day workshop October 6-7, 2009. Washington, D.C.

[29] Dempsey, Kyle B.; Brunelle, Justin F.; Jackson, G. Tanner; Boonthum, Chutima; Levinstein, Irwin B.; McNamara, Danielle S. *"MiBoard: Multiplayer Interactive Board Game".* In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[30] Derbali, L., Frasson, C. *Prediction of Players' Motivational States using Electrophysiological Measures during Serious Game Play.* ICALT 2010: 10th IEEE International Conference on Advanced Learning Technologies, Sousse, Tunisia: July 5-7, 2010

[31] Diller, D. E., Ferguson, W., Leung, A. M., Benyo, B., & Foley, D. (2004). Behavior Modeling in Commercial Games. In Papers and presentations from the 13th brims conference.

[32] Koedinger, Kenneth R. Aleven, Vincent. Heffernan, Neil. McLaren, Bruce. Hockenberry, Matthew. *"Opening the Door to Non-Programmers: Authoring Intelligent Tutor Behavior by Demonstration"*. Intelligent Tutoring Systems, Vol. 3220, pg. 7-10 (2004).

[33] Dragon, T., Woolf, B. P., Marshall, D. and T. Murray. *"Coaching within a domain independent inquiry environment."* Proceedings of the Fifth International Conference on Intelligent Tutoring Systems. Jhongli, Taiwan, Springer 4053, 144-153 (2006)

[34] Dragon, T., Floryan, M., Woolf, B.P., Murray, T. ***"****Recognizing Dialogue Content in Student Collaborative Conversation.****"*** In Proceedings of the 10th International Conference for Intelligent Tutoring Systems, ITS 2010. Pittsburgh, PA (2010).

[35] Dragon, T., Woolf, B.P. *"Understanding and Advising Students from within Inquiry Tutors."* Proceedings of the Workshop for Inquiry Learning of the 13th International Conference of Artificial Intelligence in Education. AIED 2007. Marina Del Rey, CA.

[36] Easterday, M.W.: Policy World: A cognitive game for teaching deliberation. In: Pinkwart, N., McLaren, B. (eds.) Educational technologies for teaching argumentation skills. Ben- tham Science Publ., Oak Park (in press)

[37] Easterday, M.W., Aleven, V., Scheines, R.: The logic of Babel: Causal reasoning from con- flicting sources. In: Proc. ITS for Ill-Defined Domains Workshop. AED 2007, Los Angeles, USA (2007)

[38] Fromme, Johannes. *"Computer Games as a Part of Children's Culture"*. The International Journal of Computer Game Research. Volume 3, Issue 1. May 2003.

[39] Games with a Purpose. www.gwap.com/gwap Last accessed on 2 February 2012.

[40] Gee, James. *"What Video Games Have to Teach Us About Learning and Literacy."* Palgrave MacMillan 2007

[41] Gee, James Paul. *"Learning by Design: good video games as learning machines"* E-Learning, Volume 1, Number 1. 2005.

[42] Google Image Labeler. Copyright Google inc. 2006 - 2011

[43] Greer, Jim; McCalla, Gordon; Cooke, John; Collins, Jason; Kumar, Vive; Bishop, Andrew; Vassileva, Julita. "*The Intelligent Helpdesk: Supporting Peer-Help in a University Course*" Proceedings of the 4th International Conference on Intelligent Tutoring Systems. London, UK 1998.

[44] Hallinen, Nicole; Walker, Erin; Wylie, Ruth; Ogan, Amy; Jones, Christopher. "*I Was Playing When I Learned: A Narrative Game for French Aspectual Distinctions"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[45] Hastings, Peter; Britt, Anne; Sagarin, Brad et al... "*Designing a Game for Teaching Argumentation Skills"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[46] Hodhod, Rania; Kudenko, Daniel; Cairns, Paul "*AEINS: Adaptive Educational Interactive Narrative System to Teach Ethics"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[47] Ito, Mizuko. *"Sociocultural Contexts of Game-Based Learning"* In Committee for Learning Science: Computer Games, Simulations, and Education 2-day workshop October 6-7, 2009. Washington, D.C.

[48] Johnson, W. Lewis; Vilhjalmsson, Hannes; Marsella, Stacy. *"Serious Games for Language Learning: How much Game? How much AI?"*. In Proceedings of the 2005 Conference on Artificial Intelligence in Education. Amsterdam, Netherlands.

[49] Kafai, Yasmin B. *"Playing and Making Games for Learning: Instructionist and Constructionist Perspectives for Game Studies"*. *Games and Culture;* 1 (January 2006): pg. 36-40.

[50] Katelhut, Diane Jass. *"Rethinking Science Learning: A Needs Assessment"*. In Committee for Learning Science: Computer Games, Simulations, and Education 2-day workshop October 6-7, 2009. Washington, D.C.

[51] Kelly, Henry; Howell, Kay; Glinert, Eitan; Holding, Loring; Swain, Chris; Burrowbridge, Adam; Roper, Michelle. *"How to Build Serious Games"*. Communications of the ACM – Creating a Science of Games. Volume 50, Issue 7, July 2007.

[52] Kirriemuir, John. *"Video Gaming, Education, and Digital Learning Technologies: Relevance and Opportunities"*. In D-Lib Magazine: Volume 8 Number 2. February 2002.

[53] Kittur, Aniket; Chi, Ed H. Suh, Bongwon. *"Crowdsourcing User Studies With Mechanical Turk"*. Proceedings of the ACM CHI Conference. Florence, Italy. 2008.

[54] Klopfer, Eric; Perry, Judy; Squire, Kurt; Jan, Ming-Fong; *"Collaborative learning through augmented reality role playing"*. Proceedings of the 2005 Conference on Computer Support for Collaborative Learning. Learning 2005.

[55] Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education, 8,* 30-43.

[56] Koedinger, K. R., Corbett, A. T., Ritter, S., & Shapiro, L. (2000). Carnegie Learning's Cognitive Tutor™: Summary Research Results. White paper. Available from Carnegie Learning Inc., 1200 Penn Avenue, Suite 150, Pittsburgh, P A 15222, E-mail: info@carnegielearning.com, Web: http://www.carnegielearning.com

[57] Laird, John E. van Lent, Michael. *"The Role of AI in Computer Game Genres"*. (2000) http://ai.eecs.umich.edu/people/laird/papers/book-chapter.htm

[58] Lane, H.C.**,** Hays, M.J., Auerbach, D., & Core, M.G.. *"Investigating the relationship between presence and learning in a serious game."* In J. Kay & V. Aleven (Eds.), Proceedings of the10th International Conference on Intelligent Tutoring Systems, Springer.

[59] Linton, Frank. Schaefer, Hans-Peter. *"Recommender Systems for Learning: Building User and Expert Models through Long-Term Observation of Application Use"*. User Modeling and User-Adapted Interaction. Volume 10. Pages 181-207. 2000.

[60] Lombard, M., Ditton, T.: *"At the Heart of It All: The Concept of Presence."* Journal of Computer-Mediated Communication 3 (1997)

[61] *"Math Blaster"* From KnowledgeAdventure.com 2008. Accessed on Nov. 18, 2009. http://www.knowledgeadventure.com/mathblaster/

[62] McAlinden, Ryan; Gordon, Andrew S. Lane, H. Chad; Pynadath, David. "*UrbanSim: A Game-based Simulation for Counterinsurgency and Stability-focused Operations"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[63] McKenzie, Adam; McCalla, Gord. *"Serious Games for Professional Ethics: An Architecture to Support Personalization"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[64] Mitrovic, Antonija; Mayo, Michael; Suraweera, Pramuditha; Martin, Brent; *"Constraint-Based Tutors: A Success Story"*.

[65] Mitrovic, Antonija; Suraweera, Pramuditha; Martin, Brent; Zakharov, Konstantin; Milik, Nancy; Holland, Jay *"Authoring Constraint-Based Tutors in ASPIRE"*.

[66] Murray, T., Woolf, B., & Marshall, D. (2004). Lessons learned from authoring for inquiry learning: A tale of authoring tool evolution. In J. C. Lester, R. M. Vicari, & F. Paraguaçu (Eds.), Proceedings of the 7th International Conference on Intelligent Tutoring Systems (ITS 2004) (pp. 197–206). Berlin: Springer.

[67] Murray, T., Blessing, S., & Ainsworth, S. (Eds.) (2003). Authoring Tools for Advanced Technology Learning Environments: Towards cost-effective adaptive, interac- tive and intelligent educational software. Dordrecht, The Netherlands: Kluwer.

[68] Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. International Journal of Artificial Intelligence in Education, 10, pp. 98-129.

[69] Nelson, Brian; Ketelhut, Diane Jass; Schifter, Catherine. *"Embedded Assessments of Science Learning in Immersive Educational Games: The SAVE Science Project"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[70] Noronha, Jon; Hysen, Eric; Zhang, Haoqi; Gajos, Krzysztof Z. *"PlateMate: Crowdsourcing Nutrition Analysis from Food Photographs"*. Proceedings of the 24th annual ACM symposium on User interface software and technology. New York, NY (2011)

[71] Oracle Corporation. *"Ensuring Web Service Quality for Service-Oriented Architectures"*. An Oracle White Paper. February 2010.

[72] Pinelle, David; Wong, Nelson; Stach, Tadeusz; *"Heuristic Evaluation for Games: Usability Principles for Video Game Design"*. Proceedings of the 26th SIGCHI Conference on Human Factors in Computing Systems. Florence, Italy. 2008.

[73] Plass, Jan L. Homer, Bruce D. Hayward, Elizabeth O. *"Design Factors for Educationally Effective Animations and Simulations"*. Journal of Computing in Higher Education. Volume 21, Number 1, April 2009. Pgs: 31-61.

[74] Prensky, Marc, "*Digital Game-Based Learning"*. McGraw-Hill, 2001.

[75] Rai, Dovan; Heffernan, Neil; Gobert, Janice; Beck, Joseph. "*Mily's World: Math game involving authentic activities in visual cover story"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[76] P. Wouters, E.D. Van der Spek, and H. Van Oostendorp, *"Current practices in serious game research: a review from a learning outcomes perspective"*. In T. M. Connolly, M. Stansfield & L. Boyle (Eds), Games-based learning advancements for multisensory human computer interfaces: techniques and effective practices (pp. 232–255). Hershey, PA: IGI Global. 2009.

[77] Rowe, Jonathan P. Mott, Bradford W. McQuiggan, Scott W. et al... "*Crystal Island: A Narrative-Centered Learning Environment for Eighth Grade Microbiology"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[78] Sanchez, J., Mendoza, C., & Salinas, A. (2009). Mobile serious games for collaborative problem solving. *Studies in Health Technolology and Informatics, 144*, 193-197.

[79] Schaller, D. 2006. *What Makes a Learning Game?* DOI=http://www.eduweb.com/schaller-games.pdf

[80] Shute, Valerie; Ventura, Matthew; Bauer, Malcolm; Zapata-Rivera, Diego. *"Melding the Power of Serious Games and Embedded Assessment to Monitor and Foster Learning: Flow and Grow"*. In The Social Sciences of Serious Games: Theories and Applications (2009).

[81] Shute, V. J. *"Stealth assessment in computer-based games to support learning"*. S. Tobias & J. D. Fletcher (Eds.), *Computer games and instruction*. Charlotte, NC: Information Age Publishers.

[82] Shute, V. J., & Kim, Y-J. *"Does playing the World of Goo facilitate learning?*" In D. Y. Dai (Ed.), *Design research on learning and thinking in educational settings: Enhancing intellectual growth and functioning*. New York, NY: Routledge Books.

[83] Siem, Knut Vidar. *"Artificial Intelligence in Computer Games"*. College of Staten Island, City University of New York. May 16, 2006

[84] Siorpaes, Katharina and Martin Hepp. Ontogame: Towards overcoming the incentive bottleneck in ontology building, 2007.STI TR 2011-05-01 19

[85] Siorpaes, Katharina and Martin Hepp. Games with a purpose for the semantic web. IEEE Intelligent Systems, 23(3):50–60, 2008.

[86] Siorpaes, Katharina and Martin Hepp. Ontogame: Weaving the semantic web by online games. In ESWC, pages 751–766, 2008.

[87] Siorpaes, Katharina and Martin Hepp. Ontogame: Weaving the semantic web by online games. Springer LCNS, 2008.

[88] Siorpaes, Katharina and Elena Simperl. Human intelligence in the process of semantic content creation. World Wide Web (WWW) Journal, 13(1), 2010.

[89] *"SimCity"* From SimCitySocieties.ea.com 2008. Accessed on Dec. 3, 2009. http://simcitysocieties.ea.com/index.php

[90] Suthers, D. D. (2003). Representational guidance for collaborative inquiry. In J. Andriessen, M. J. Baker, & D. D. Suthers (Eds.), Arguing to learn: Confronting cognitions in computer-supported collaborative learning environments (pp. 27–46). Dordrecht: Kluwer Academic.

[91] Suthers, D. D., & Hundhausen, C. (2003). An experimental study of the effects of representational guidance on collaborative learning processes. Journal of the Learning Sciences, 12(2), 183–219.

[92] Suthers, D. D., Weiner, A., Connelly, J., & Paolucci, M. (1995). Belvedere: Engaging students in critical discussion of science and public policy issues. In J. Greer (Ed.), Proceedings of the 7th World Conference on Artificial Intelligence in Education (AI-ED 1995) (pp. 266–273). Charlo ttesville: Association for the Advancement of Computing in Education.

[93] Tecuci, Gheorghe; Keeling, Harry. *"Developing an Intelligent Educational Agent with Disciple"* Artificial Intelligence in Education, 10, 1999, 221-237.

[94] Thinking Worlds. Rapid Sims and Games Creation. Caspian Learning (2011). www.thinkingworlds.com

[95] Thomas Markotschi and Johanna Volker. Guess what?! human intelligence for mining linked data. 2010.

[96] Thomas, James M. Young, R. Michael. "*Dynamic Guidance in Digital Games"*. In Proceedings of the Workshop on Intelligent Educational Games 2009. Brighton, England.

[97] Tuzun, Hakan; Yilmaz-Soylu, Meryem; Karakas, Turkan; Inal, Yavuz; Kizikaya, Gonca; *"The effects of computer games on primary school students' achievement and motivation in georgraphy learning"*. Computers and Education. Volume 52, Issue 1, Pages 68-77. January 2009.

[98] Van Eck, R. (2006) Digital game-based learning: It's not just the digital natives who are restless. *EDUCAUSE review*, March/April, 16-30

[99] Von Ahn, Luis; Dabbish, Laura. *"Designing Games with a Purpose"*. Communications of the ACM. Volume 51, Issue 8, August 2008.

[100] Vassileva, Julita; McCalla, Gordon; Greer, Jim. "*Multi-Agent Multi-User Modeling in I-Help*". Journal of User Modeling and User-Adapted Interaction. Volume 13, Issue 1-2, February-May 2002.

[101] *"Video Game Industry Statistics"* From GrabStats.com 2008. Accessed on Nov. 14, 2009. http://www.grabstats.com/statcategorymain.asp?StatCatID=13

[102] Wang, Aobo. Hoang, Cong Duy Vu. Kan, Min-Yen. *"Perspectives on Crowdsourcing Annotations for Natural Language Processing*" (2010)

[103] Woolf, "*Building interactive intelligent tutoring systems"*, Morgan Kaufmann 2008.

[104] Woolf, B.P., Murray, T., Marshall, D., Dragon, T., et al., *"Critical Thinking Environments for Science Education"*. Proceedings of the 12th International Conference on AI and Education, (2005)

[105] Yen-ling Kuo, Jong-Chuan Lee, Kai-yang Chiang, Rex Wang, Edward Shen, Cheng-wei Chan, and Jane Yung-jen Hsu. Community-based game design: experiments on social games for commonsense data collection. In Proceedings of the ACM SIGKDD Workshop on Human Computation, HCOMP '09, pages 15–22, New York, NY, USA, 2009. ACM.

[106] Apache Lucene Core V. 3.6.1; http://lucene.apache.org/core/

[107] Bloom B.S., *"The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring"* Educational Researcher, vol. 13, no. 6, p. 4, Jun. 1984.

[108] Cohen P.A., Kulik J.A., and Kulik C.-L. C., *"Educational Outcomes of Tutoring: A Meta-analysis of Findings,"* American Educational Research Journal, vol. 19, no. 2, pp. 237–248, Jun. 1982.

[109] Floryan, M., Dragon, T., Woolf, B.P. ***"****When Less is More: Focused Pruning of Knowledge Bases to Improve Recognition of Student Conversation "* In Proceedings of the 11th International Conference for Intelligent Tutoring Systems, ITS 2012. Chania, Greece (2012).

[110] Floryan, M. Woolf, B.P. *"Students that Benefit from Educational 3D Games"*. Proceedings of the IEEE International Conference on Advanced Learning Technologies. Athens, GA (2011)

[111] Fletcher, J. D. *"Does this stuff work? Some findings from applications of technology to education and training."* Conference on Teacher Education and the Use of Technology Based Learning System. Warrenton, VA (1996). Society for Applied Learning Technology.

[112] Regian, J. W. *"Functional area analysis of intelligent computer-assisted instruction"* (Report of TAPSTEM ICAI-FAAA Committee). Brooks AFB, TX (1997).

[113] Van Lehn, K., Lynch, C., Schulze, K., Shapiro, J.A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., and Wintersgill, M. (2005). The Andes Physics Tutoring System: Lessons Learned. *International Journal of Artificial Intelligence and Education*, 15 (3).

[114] Van Lehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R. H., Taylor, L., Treacy, D. J., Weinstein, A., and Wintersgill, M. C. (2005). The Andes physics tutoring system: Five years of evaluations. In: G. I. McCalla and C.-K. Looi (Eds.), *Proceedings of the Artificial Intelligence in Education Conference*. Amsterdam: IOS.

[115] Clarke, J., Dede, C., Ketelhut, D. J., Nelson, B., & Bowman, C. (2006). *Multiuser virtual environments (MUVEs) as research tools to assess student learning.* Paper presented at the American Educational Research Association, San Francisco, CA.

[116] Johnson, W. Lewis *"A Simulation-Based Approach to Training Operational Cultural Competence"* In Proceedings of ModSIM 2009. March 2010.

[117] Dragon, T. *"The Impact of Integrated Coaching and Collaboration within an Inquiry Learning Environment"*. Doctoral Dissertation. University of Massachusetts, Amherst (2013).

[118] Ha, E. Y., Rowe, J. P., Mott, B. W., & Lester, J. C. (2011). Goal recognition with Markov Logic Networks for player-adaptive games. *Proceedings of the 26th National Conference on Artificial Intelligence* (pp. 2113–2119).

[119] Meluso, A., Zheng, M., Spires, H., & Lester, J. (2012). Enhancing 5th graders' science content knowledge and self-efficacy through game-based learning. *Computers and Education*, *59*(2), 497–504.

[120] Rowe, J. P., Shores, L. R., Mott, B. W., & Lester, J. C. (2011). Integrating learning, problem solving, and engagement in narrative-centered learning environments. *International Journal of Artificial Intelligence in Education*, *21*, 115–133.

[121] Shores, L. R., Rowe, J. P., & Lester, J. C. (2011). Early prediction of cognitive tool use in narrative-centered learning environments. *Proceedings of the Fifteenth International Conference on Artificial Intelligence in Education*. Auckland, New Zealand.

[122] Aleven, V., McLaren, B. M., & Sewall, J. (2009). Scaling up programming by demonstration for intelligent tutoring systems development: An open-access website for middle-school mathematics learning. *IEEE Transactions on Learning Technologies, 2*(2), 64-78.

[123] Aleven, V., McLaren, B.M., Sewall, J., & Koedinger, K.R. (2009). A New Paradigm for Intelligent Tutoring Systems: Example-Tracing Tutors. *International Journal of Artificial Intelligence in Education, 19*(2), 105-154.

[124] Mitrovic, A., Martin, B., Suraweera, P., Zakharov, K., Milik, N., Holland, J., & McGuigan, N. (2009). ASPIRE: an authoring system and deployment environment for constraint-based tutors. *International Journal of Artificial Intelligence in Education,* 19, 155-188.

[125] Mitrovic, A., McGuigan, N., Martin, B., Suraweera, P., Milik, N., Holland, J. (2008). Authoring constraint-based tutors in ASPIRE: A case study of a capital investment tutor. In *Proceedings of theWorld Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA 2008)* (pp. 4607-4616). Chesapeake, VA: AACE.