# Custom Feedback Selection for Intelligent Tutoring Systems in Ill-Defined Domains

by

Stuart Henry Johnson

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

# Abstract

Current medical imaging professional training uses an apprenticeship model with students following an established doctor and viewing their cases, in what is called a practicum. This posses an issue as students are limited to the cases available during their practicum. To resolve this automated instruction can aid in their education promoting both increased depth and breadth. To accomplish this we have created a new Intelligent Tutoring System, Shufti. Shufti makes use of modern gamification and Intelligent Tutoring System designs to augment the learning experience of mammography students. In Shufti we have introduced a new reinforcement learning based technique for use in Intelligent Tutoring System feedback selection in ill-defined domains, and have made use of modern gamification techniques to increase learner engagement.

*To my family and loved ones*

*For always believing in me.*

*I think there is a world market for maybe five computers.*

– Thomas J. Watson, IBM Chairman, 1943.

# Acknowledgements

Thank you to my loving family you always supported me. Thank you to Osmar Zaiane for pushing me to be the best student I could. Thank you to Brittany Nouwen for all the time you spent reading this and listening to me.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

Analysis of medical images is an important tool in modern medicine. Modern technology has changed how we explore and diagnose issues. Where once it would be necessary to perform a risky exploratory surgery to determine the source of an ailment, now an MRI can peer into the depths of the human body from outside. Beyond simply making diagnosis easier, modern technology has enabled remote diagnosis; an expert no longer needs to be present to provide insight.

These advances, though, come with a downside: the volume and variety of medical images produced has increased by leaps and bounds. With technologies such as Magnetic Resonance Imaging (MRI) becoming common place, the demand for professionals able to analyze them has increased. Unfortunately the current process of training such professionals follows an apprentice model with medical students following a fully practicing doctor in what is known as a practicum, during which they assist their instructor and review their cases. This has proven historically adequate but has some downsides. The students are only exposed to the cases that their instructor receives meaning that rare conditions are not seen, resulting in a less than comprehensive education. In addition, the limited number of cases seen by a doctor mean that the students have a limited ability to practice. Further worsening this is the difficulty of analyzing medical images, resulting in poor rates of detection [21].

## 1.2 Problem Definition

An Intelligent Tutoring System could help with these issues in a variety of ways. By increasing the students access to the cases including rare ones it can improve the breadth of their experience. To induce practice one can use techniques borrowed from serious games [30]. To retain the quality of instruction from tutoring [4] that a student receives during their practicum such an environment can use intelligent tutoring techniques.

To this end we have created a medical imaging Intelligent Tutoring System, Shufti. Shufti is a mammography Intelligent Tutoring System. Mammography was chosen as the focus of Shufti over other medical imaging fields due to the availability of data and the pressing need for more professionals able to perform analysis of mammograms. It uses gamification and social techniques to provide the learner with motivation to learn. To increase students access to cases it draws cases from large existing databases of medical images. To retain the benefits of tutoring it makes use of innovative student modelling to provide adaptive learner specific feedback in a domain of study which, due to its structure does not lend itself to normal techniques.

Serious games, games which are used to teach serious topics, are an up and coming innovation in education [28][24]. Gamification has many benefits such as being habit forming [35], resulting in more practice. Completion can transform routine tasks into interesting challenges. Social features can further improve the learning experience by allowing learners to share knowledge amongst one another. Our implementation of gamification techniques is explored in Section 6.3.

There are many large medical image databases online from which is it possible to draw exercises. Some examples are the Digital Database for Screening Mammography [13] or the Segmentation Chest Radiograph Database [31]. This enables students to see a broader set of cases than they normally would. These databases also contain rare cases as well, allowing students to increase the depth of the cases that they have seen by emphasizing rare or difficult cases. Our use of these is outlined in the introduction to Section 6.

An Intelligent Tutoring System can provide leaner specific adaptive feedback to the students, simulating such a tutoring experience. In Shufti's case a innovative form of Reinforcement Learning [29] based student modelling is used to provide advice and feedback to the leaner. This was done due to medical imaging being what is know as an ill-defined domain which is a domain which does not lend itself to normal Intelligent Tutoring System techniques. The motivation, challenges, methods, and validation of this are outlined in Section 7.1.

Adaptive feedback can be achieved via many means, such as ELM-PE's automatic presentation of relevant information to the learner [33] or automatically and pre-emptively pointing out mistakes as seen in The Logic Tutor: A Multimedia Presentation [1]. It is also possible to guide the learner along the discrete steps necessary for solving a task as shown in the ITS PETE [12]. Unfortunately though, such techniques are inadequate for use in Shufti as it is a mammography ITS.

Mammography is known as an ill-defined domain, which means it assesses problems which do not have a definitive answer and require the relation of diverse, yet relevant concepts to solve them [18]. In this dissertation we will be exploring the topic of Intelligent Tutoring Systems, with emphasis on feedback and ill-defined domains. Given that conventional techniques for constructing Intelligent Tutoring Systems are not effective in an ill-defined domain, this study will focus on how Shufti processes and delivers feedback within this framework to successfully provide training to its learners.

## 1.3 Thesis Statement

We have created a system that provides individualized adaptive feedback for Intelligent Tutoring Systems in ill-defined domains.

Feedback selection in ill-defined domains is difficult for a variety of reasons, most prominently due to ill-defined domains being more difficult for computers to work in. For example, creative writing is unsuited to automatic reasoning making it difficult to automatically issue effective feedback.

## 1.4 Thesis Contributions

Put more specifically, the central contribution of this document is the introduction of a new reinforcement learning based technique for selecting custom, learner specific, feedback in ill-defined domains. Shufti, which was developed concurrently with this study, uses this innovation to provide feedback to mammography learners. Shufti also makes use of modern gamification techniques so as to improve learner use of the system. Additionally this document contains the design and structure of a learner simulator with which it is possible to evaluate a feedback issuing system.

## 1.5 Research Methodology

We have created Shufti a new ITS, focused on mammography. In which we employ a new method of selecting feedback to issue to learners. Alongside that we have also incorporated gammifciation techniques to further improve its effectiveness as a learning aid.

To validate the performance of the new feedback selection method in Shufti it would have been desirable to make use of two courses teaching the diagnosis of mammograms, one with access to the system and one without, for comparison. Unfortunately we do not have access to such classes because of the small practicum groups. We have instead created a realistic simulation of a mammography course, with simulated learners that learn and react to feedback in a realistic individual manner.

## 1.6 Document Structure

The content and structure of this dissertation is as follows: Chapter 2 is an examination of the topic of Intelligent Tutoring Systems. It explores both the definition of important concepts for the field, such as what is feedback, hint, a domain model, a student model and a tutoring model.

In Chapter 3 we focus on what is a well-defined domain. What attributes does a domain have to have to be well-defined. What is a typical structure of

an ITS in such a domain. It also talks provides examples of existing ITSs in well-defined domains.

In Chapter 4 we define what is an ill-defined domain in contrast with well-defined domains. We then provides examples of existing ITSs in ill-defined domains and the difficult of validating them.

Chapter 5 starts with a definition of feedback and its possible attributes. We then provide an enumeration of each of these possible feedbacks so as to explore the possibilities. Finally, we examine some existing methods of providing feedback in ill-defined domains.

In Chapter 6 we introduce our new mammography ITS Shufti. We examine its features, design, and structure.

Chapter 7 is a summary of how feedback in Shufti is provided via reinforcement learning; including a definition of what reinforcement learning is.

Chapter 8 begins with the design of a simulator for validating Shufti's feedback system. We then describe the exercises, courses, and learners within this simulator. We then discuss the simulators used to select the reinforcement learning algorithm, and parameters for Shufti's feedback system and then concludes with a discussion of the results of the evaluation.

Chapter 9 is an examination of other possible fields to which the techniques used in Shufti can be used.

# Chapter 2

# ITS Structure and Features

An Intelligent Tutoring System (ITS) is educational software which recreates the one-on-one human tutoring experience by choosing effective pedagogical techniques and stratagems to improve the learner's performance. This is done to simulate the effectiveness of a human tutor as described in Bloom's "The 2 Sigma Problem" [4], in which the testing showed that 90% of tutored students achieved the same level of mastery as the top 20% of conventionally instructed students.

ITSs target many fields, or as we will refer to them domains. A domain is a field of instruction which can be as broad as arithmetic or as narrow as how to repair a particular device. A domain can often stretch beyond the normal fields of instruction such as the training of military officers [6].

Intelligent Tutoring Systems attain their goal of simulating tutoring through various means such as providing advice, guidance and information to the learner. This can take the form of a simple score presented after an exercise is completed, to advice and illustration of domain principles violated by a user's answer.

## 2.1 Hints and Feedback

Hints and Feedback are extremely important concepts in intelligent tutoring. As a rule, in this manuscript, Hints are information that the user requests specifically (for example clicking a button or following a link in an ontology) and Feedback is unrequested information selected by the system to present to

the learner, such as notifying them of a mistake.

## 2.2 Conventional ITS Structure

Though the focus of this document is on ITSs in ill-defined domains, it is perhaps prudent to summarize the structure of a conventional Intelligent Tutoring System, in a well-defined domain. A conventional ITS structure is as follows: a domain model, a student model, a tutoring model, and a user interface [23]. The following sections examine each one of these models in turn, briefly touching on some common techniques used in their design.

### 2.2.1 Domain Model

The domain model is the "understanding" of the field in which an ITS will tutor; it provides the system the ability to reason using domain principles. For an intuitive metaphor, a domain model can be thought of as an instructor's knowledge of the subject matter they are teaching. They would use this knowledge to validate the steps taken by the learner they are tutoring and to suggest exercises and provide hints. There are many possible types of domain models [22], though the majority fall under the label of expert systems.

Nkambou [22] defines expert system domain modelling as using an expert system to reason about the domain in question. An expert system is a problem-solving system which uses production rules to represent the dynamics of its field. Nkambou further states that from an intelligent tutoring perspective, there are three kinds of expert systems: Cognitive models in which the knowledge representation and the reasoning method employed are understandable to humans, Glass Box models in which the knowledge representation is understandable to humans but the reasoning mechanism is not, and Black Box models in which neither the knowledge representation or reasoning are understandable. They vary in usefulness and power based on their transparency, with the most transparent, Cognitive models, able to offer intermediate steps to the learner during problem-solving, and the least transparent, Black box models, only able to validate the learner's actions and answers. Two other ex-

amples of domain models can be found in chapters of "Advances in Intelligent Tutoring Systems" [23]. First, in Rule-Based Cognitive Modeling for Intelligent Tutoring Systems [2] Aleven explores how a production-rule based expert system can be used in an ITS. Second, in "Modeling Domains and Students with Constraint-Based Modeling" [20] Mitrovic proposes the application of constraints to the system to maintain the domain model's consistency instead of giving the system the ability to reason about the domain.

### 2.2.2 Student Model

A student model is the conception of a student's knowledge and emotional state as outlined in Woolf [34]. Woolf states "A student model in an intelligent tutor observes student behaviour and creates a qualitative representation of their cognitive and affective knowledge".

There are many possible techniques for student modelling. One can store student proficiencies, ranging from simple numeric rankings, to complex networks of interrelated skills. Many student models will learn rules about what the student does and does not know, while other's will attempt to place students into stereotypical categories of learners.

### 2.2.3 Tutoring Model

Tutoring models as described in Bourdeau and Grandbastien [5] are the sections of the ITS devoted to selecting pedagogical strategies, such as selecting exercises for the learner to perform and what feedback of hints to provide. They do this by incorporating information from the student model and the domain model. They range in complexity from simple lists of exercises done in order to complex systems which select exercises based on predictions of student performance and educational gains.

# Chapter 3

# ITS Examples in Well-Defined Domains

In this chapter we will explore what a well-defined domain is and examine three Intelligent Tutoring Systems which have been successful in educational contexts. First is a web based Lisp programming ITS called ELM-PE, which stands for Episodic Learner Model - Programming Environment, created by Weber and Mollenberg [33], chosen as an example of how the framing of a problem can influence how well-defined its domain is. Second is Logic Tutor[1] which is a Java based symbolic logic ITS by Abraham et al, chosen as it is an excellent example of an ITS in a well-defined domain. Third we will explore ActiveMath, a mathematics ITS [19], chosen as it is both an ITS in a well-defined domain and is in use in real world classrooms. Lastly we will touch on some other interesting ITSs and developments.

## 3.1 Well-defined Domains

A domain is called well-defined if it has all of the following attributes: verifiability, formal theories, a well-defined task structure, clearly defined concepts, and a decomposable task structure [10]. These attributes also enable the use of conventional intelligent tutoring techniques and domain modelling. In this section we will examine each one of the attributes and indicate why it is necessary.

### 3.1.1 Verifiability

A well-defined domain must have verifiable answers - such as in arithmetic where there is only one correct answer for a given problem - or, at the very least, the ability to distinguish a correct answer from an incorrect answer. Problems in ill-defined domains lack verifiability in that they commonly do not have a truly correct answer. For example, consider a writing exercise in which the goal is to write an interesting story. It is possible to write an interesting story and it is possible for one to manually determine if a story is interesting, but there is no one correct answer to the exercise "write an interesting story". In other words, an ambiguous notion of correctness is one of the possible features of an ill-defined domain, which is in conflict with the idea of verifiability.

### 3.1.2 Formal Theories

A well-defined domain must have a clear cut set of rules, or formal theorems within which one can reason and solve problems. An example of the use of formal theorems is symbolic logic in which the entire problem space is well-defined and there are rules which can be applied in all situations. Contrast this with the case of performing a medical diagnosis. In medicine there are only vague notions of formal theories, such as the effects of a drug on a disease. This vagueness and lack of theories frustrates the creation of domain models which results in ill-defined domains. It is interesting to note that generally when one has fully fleshed out domain rules one also gains the attribute of verifiability but the opposite is not always true.

### 3.1.3 Well-Defined Task Structure

A well-defined domain must have a regular task structure or, as stated in Fournier-Viger et al. [10], be a "Problem Solving Domain". A problem solving domain is one in which there is one correct answer, where given enough information such an answer can be determined, without the use of the solver's judgement. This contrasts with other structure such as design or analytics. In

the case of design, novelty is the desired meaning that there is no one correct answer. In the case of an analytical domain the learner is required to use their judgement which results in any answer being ambiguous. Another notion of this is idea that a domain has a well defined task structure if it is possible to automatically generate exercises with in it. This is due to the regularity in domain tasks and subject matter necessary to generate them in a realistic manner.

### 3.1.4 Clearly defined concepts

For a domain to be well-defined, the concepts within it must have a clear definition, without ambiguity. For example, a triangle is a well-defined concept with no ambiguity, something is either a triangle or not a triangle. This is not always present within a domain. For example, the various styles of art are not well-defined and a work's inclusion in a particular style is based more on consensus of the community and the statements of the artist than on any hard definition. However, clear, unambiguous definitions are necessary if one is to reason about concepts within a domain.

### 3.1.5 Decomposable task structure

For a domain to be well-defined it must be possible to decompose problems within it to a sequence of separate and independent steps. For example, when solving a logic problem, the correctness of a given step can be determined from the current state and the desired end state. The correctness of steps taken before the current step have no influence on whether or not the current choice is correct. Contrast this with designing an engine, a problem which is composed of many overlapping sub-problems where the solution to one affects the correct solution to the others. As an example, let us assume an engineer has to design an engine to meet specific requirements for dimensions, output and weight. Not only does the engineer have to decide how many cylinders are required but this then, in turn, affects the dimensions of the cylinders and the weight of the engine. Because the engineer cannot decompose each of the tasks into discrete problems which do not impact other elements of

the overall solution, the number of permutations and combinations becomes unmanageable. This effectively results in what we call an ill-defined domain.

## 3.2 ELM-PE

As stated above, in the introduction to this chapter, ELM-PE [33] is a Lisp programming tutoring environment which teaches novice computing science students how to program Lisp, a non-trivial task as the system is both teaching the concepts behind computing science and the language syntax in which to communicate them. This is accomplished by providing an editor with three modes: Listener mode in which the system provides minimal assistance, only simple error messages are displayed for syntax errors; Editor mode in which the assistive (tutoring) features are enabled providing a structure of code or template for the user to complete; and lastly Exercise mode where the most assistance is provided.

In Listener mode the user is presented with a simple Lisp code editing environment in which they are told only of the syntax errors in the code they enter. This is the system's starting state and is minimally assistive, lacking any example output or guidance (templates). This can be thought of the system acting as a normal editor or IDE.

Editor mode is more assistive, providing templates for common programming patterns in Lisp which the learner is required to fill in. In this mode the learner is given significant assistance including examples of similar code and a visualization of the evaluation of their code. This system will only accept syntactically correct Lisp code and will provide explanations for why any given piece of code is incorrect. It should be noted though that the system will only provide feedback or assistance to correct code syntax. It does not provide any assistance to help ensure the code logic is correct.

Finally, Exercise mode provides a high level of assistance encompassing all of the prior levels and additionally extending its assistance and feedback to include on the fly evaluation of a program's output in order to attempt to find logical errors and to provide feedback on how to resolve them. This is done by

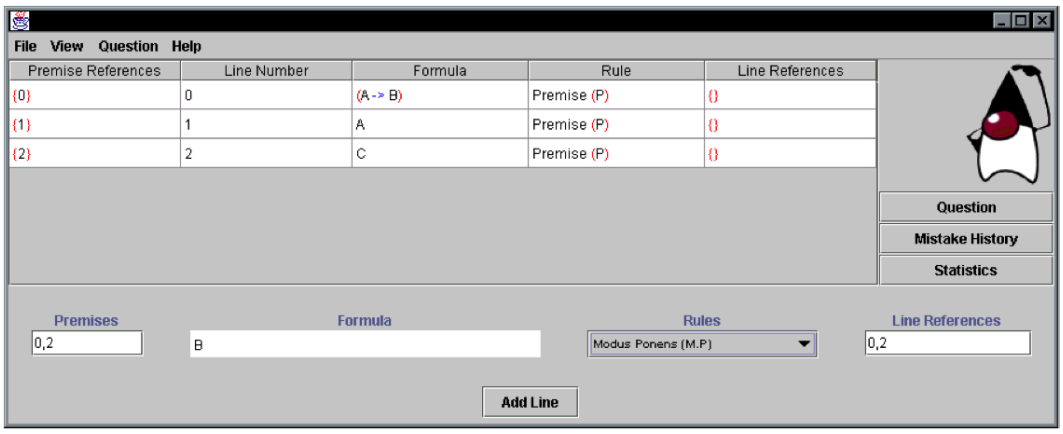


Figure 3.1: Logic Tutor's UI

comparing the student's produced output with a list of anticipated erroneous outputs. The system also makes use of a cognitive model to provide hints to help students to get "unstuck".

Overall the system appears to work quite well. Students who made use of it outperformed those who did not by 40 percentage points. There are some possible improvements to the system's feedback design; the feedback is fitted to the situation and not adapted to a particular learner. In addition the necessity of manually creating lists of wrong answers makes it time consuming to create new exercises for the system.

## 3.3 The Logic Tutor: A Multimedia Presentation

Logic Tutor is a symbolic logic tutoring environment in which students are taught how to perform formal logical proofs using symbolic logic. Symbolic logic is a field well suited to intelligent tutoring techniques as it can be fully modelled using a computer. The system takes the form of a Java application in which the students are tasked to perform step by step logical proofs.

To perform these proofs the student is provided a set of logical predicates such as $a \implies b$. Students must then enter the next transformation of the predicates, step by step, so as to prove some property. The system provides feedback to the user when they have made two mistakes. This feedback is

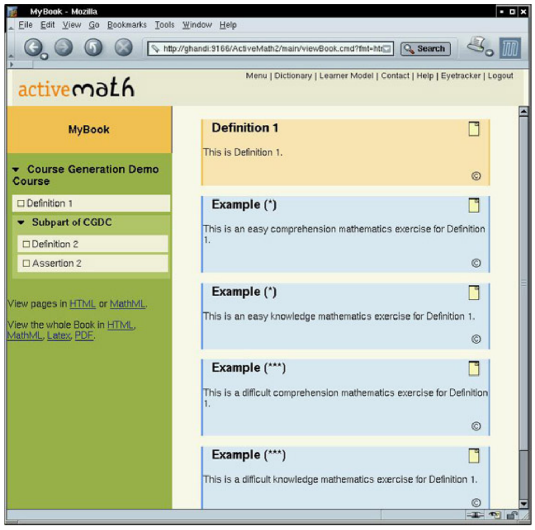


Figure 3.2: ActiveMath's UI

determined by comparing the mistakes with a set of mistake patterns, a similar process to the ELM-PE list of output errors. Errors are highlighted using colour to draw the student's attention. If multiple mistakes are made the fields of the answer are checked to see if they are filled in with the correct type of data. Next, the syntax of the formula is checked followed, then finally the correct application of the rules of symbolic logic is checked for.

The system stores user's exercise actions allowing them to be reviewed. It also has an automatic exercise generator to produce an unlimited number of possible exercises for students to practice on. These features make Logic Tutor an excellent review tool for students.

## 3.4 ActiveMath and MathTutor

ActiveMath, created by Melis [19], is a web based mathematics Intelligent Tutoring System, in which the authors aim to "support truly interruptive, exploratory learning" [19] by adaptively presenting information to the learner. ActiveMath stores mathematical principles as a set of rules as well as a list of pedagogical goals for the learner to achieve. When a learner selects a topic the system produces a custom course based on the set of rules and a modelling of the student's current competencies.

Students are modelled based on their history and the system's estimation of their competency at solving mathematical problems. When the student submits an exercise, their model is updated based on their performance on the last exercise. Interestingly, ActiveMath allows students to modify their own models to correct misconceptions that the system may have about them.

Mathematical principles within the system are represented in a semantic XML called OMDoc, which is a math principle description language. While students are performing exercises, the system uses a computer algebra system to evaluate the transformations they have used to solve the problem in order to determine if they matches one or more of the mathematical principles in the OMDoc database. If so, the student is allowed to continue. Otherwise they are notified of the error.

Another mathematics related ITS is MathTutor by Aleven et al. [3] which enables students to study mathematics subjects as varied as algebra, data analysis, and geometry. To model the domain it uses Example Tracing.

Example Tracing Tutors store a series of generalized examples of exercise solutions, and compare the actions taken by the learner with these examples. If the learner takes an action which is inconsistent with the example, the system provides feedback indicating what the error was. The system can also offer the learner hints about the next possible step in solving the problem. More detail about MathTutor can be found in "Scaling Up Programming by Demonstration for Intelligent Tutoring Systems Development: An Open-Access Web Site for Middle School Mathematics Learning" [3].

## 3.5 Other notable developments

Some additional developments in intelligent tutoring systems include: The use of Bayesian networks to model learners intentions in well defined domains [7]. Which is an interesting approach and one that has proven useful in well defined domains. Another unique ITS design approach is having the student examine their own problem solving process [8]. An idea which is similar to the replay and review functionality present in Shufti, and can be seen in Figure 6.9.

# Chapter 4

# Ill-defined Domains

The examples presented in the last chapter, Chapter 3.1, are all related to mathematics, however the techniques used in these examples cannot be applied to all domains. Examples of fields where these techniques cannot be used include writing, architecture and art. These are known as ill-defined domains.

An ill-defined domain is a domain which lacks one or more characteristics of well-defined domains namely, verifiability, formal theories, A well-defined task structure, clearly defined concepts, and a decomposable task structure as defined in Lynch et al. [18], Fournier-Viger et al. [10] and enumerated in Section 3.1.

## 4.1 Examples of ITSs in Ill-defined Domains

The following section gives examples of ITSs which are in ill-defined domains. It is important to reiterate that not every domain is equally ill-defined and how one approaches a domain can directly influence how well it is defined.

First we will examine Code Hunt [25] a computer programming ITS as it contains an interesting approach to dealing with the open-endedness of programming. We will then discus PETE [12], an engineering ethics ITS, which was chose because the field of ethics lacks almost all of the attributes of a well-defined domain from Section 3.1. And finally we will discuss SlideTutor, an ITS which focuses on Pathology as it is both relevant to later sections involving medical imaging and has an interesting approach for dealing with the lack of formal theories within pathology.


Figure 4.1: CodeHunt's UI

### 4.1.1 Code Hunt

CodeHunt [25] is a computer programming ITS which, unlike ELM-PE described in Section 3.2 , is in an ill-defined domain. The main argument for why Code Hunt is in an ill-defined domain hinges on the lack of structure in the problems the learner attempts to solve.

Learners are presented with an editor, some test cases, and a submit button. The goal is to write a small program which satisfies the test cases and then submit their program for evaluation. If the student succeeds, they are awarded points which contributes to their ranking within the system. If the student's program fails the tests, the system offers advice on how to correct the problem, which is also known as "feedback".

Because there is no enforced structure there are endless possible answers to any exercise, resulting in an ill-defined task structure. Additionally, because in programming every part of the program is interrelated the problem also contains overlapping subtasks. These two attributes - lack of a well-defined task structure and a lack of a decomposable task structure - result in the domain being ill-defined.

Due to this, the system has to employ new and innovative techniques to provide feedback to the learner. For all students it records their use of the system, including the time and the current code entered. When a learner

has not succeeded at an exercise, their current answer is clustered with other students histories. From the current answer's neighbours, the system then looks at the subsequent changes made by those who ultimately succeeded and uses that as a feedback to the learner.

### 4.1.2 PETE

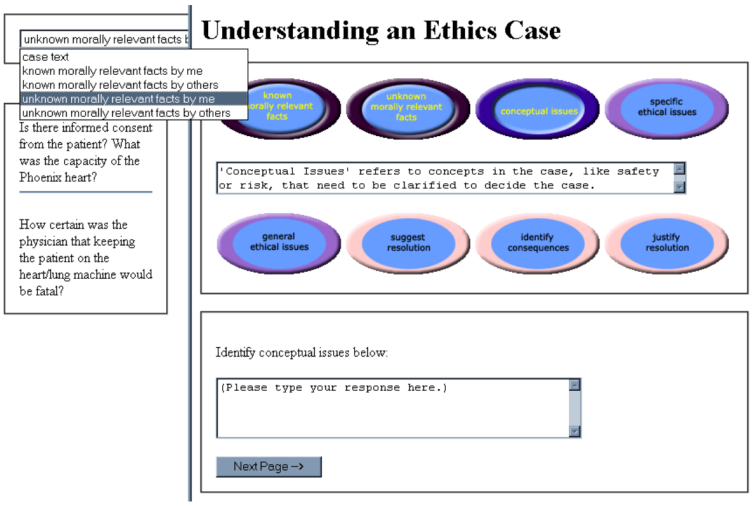


Figure 4.2: PETE's UI

In PETE by Goldin et al. [12] learners performing ethical case studies (for example a non-FDA approved artificial heart being used as a temporary replacement while a patient is waiting for a donor.)

The student is presented with an ethical case study and would first determine the ethical concepts it contains. Next, the student would identify the stake holders, and then finally the would recommend a reasonable solution to the problem. This is presented in a web-based interface where the student manually enters their answer for each step in sequence. They are able to go back and change their answers to prior questions but not to advance to later steps until they have answered the current step. This is done to enforce a common structure for the ethical dialogue.

When they have completed a step they are presented with the systems "gold standard" answer and the answers of their peers. The system has no

form of student ranking or scoring and is instead designed to facilitate the exploration of ethics by the student.

This is an excellent example of an ill-defined domain for a number of reasons. First of all there is no one correct answer to such ethical dilemmas resulting in a lack of verifiability. Secondly, ethics has no formal domain theorems with which to automatically reason. Third of all, the task structure is ill-defined as it is a judgment based task. And finally, the domain suffers from open textured concepts in that most ethical terms and ideas lack comprehensive definitions.

To deal with these problems, Goldin et al. have done two things: they have manually created definitions of ethical terms, and they have created a gold standard for what the correct answer to each step should be, in effect imposing a model on the domain. This model is not comprehensive but it is consistent with itself and current ethical thinking.

### 4.1.3 SlideTutor

SlideTutor is a pathology Intelligent Tutoring System created by Crowley et al. [9]. In it, learners are tasked with performing pathological diagnoses on tissue samples in a simulated pathology tool. The samples are presented as high resolution slides in a desktop application.

The learners select evidence for various conclusions using various annotation functionalities. They are then told if their diagnosis is correct or incorrect. During the process the students are offered automatic advice (feedback) on what the appropriate next step would be.

Pathology is an ill-defined domain for many reasons. First of all there are no formal theorems with which to reason about the domain. Secondly the task is analytical in nature and thus has an ill-defined task structure. And finally, the concepts within the field are open to interpretation and are not comprehensively defined.

To resolve this issue Crowley et al. have created an artificial model. They had expert pathologists diagnose tissue samples while creating annotations of their reasoning and highlighting the regions they are referring to. From this

information they produced a model of diagnosis for each exercise providing the system with a list of possible actions at each step of the process.

## 4.2 Validation of ITSs in Ill-defined Domains

Validating an ITS is a difficult task. One requires both a means to determine if the system is functioning correctly and a way to determine if its pedagogical goals have been met.

Empirical validation of a system's performance on students is the most common approach to validation (as was done with ActiveMath and MathTutor) however this requires finding a body of students to perform the validation with.

Unfortunately it is often difficult to find a suitable group of students, and as a consequence many system remain unvalidated. Self [27] has argued that too many validations are focused on the pedagogical approach, and neglect to show if the system will work properly in general. Some systems have followed Self's advice and have adopted a general validation approach, such as with CodeHunt where they do not examine the learning gains made by the users but simply evaluate if the system has issued a useful feedback.

Our own solution to this problem can be found in Section 8 where we propose a compromise between the empirical approach and the theoretical validation approach.

# Chapter 5

# Feedback

In this next section we will describe feedback as a concept in ITS. We will comprehensively categorize possible feedback strategies in order to fully explore the problem space.

It is important to note that many of the feedback structures proposed in the following sections are commonly referred to as hints, but we have chosen to make a distinction. Hints are information that the user has requested by performing an action such as pressing a hint button. Feedback is information that the tutor provides in an unprompted manner, produced as a side effect of the learners actions.

## 5.1 Feedback: a Definition and Exploration

Feedback is an important concept in Intelligent Tutoring Systems. It is information provided to the learner in an unprompted manner by the system. Feedback has three main attributes: Timing, Polarity and Specificity.

Timing of feedback determines where and when the learner will receive new information from the system. It falls into three categories: Pre-exercise Feedback, Post-Exercise Feedback and Concurrent Feedback.

Pre-exercise feedback is feedback which occurs before an exercise is performed such as presenting the relevant concepts for an exercise before the learner attempts it.

Post-Exercise feedback is feedback taking place after the learner has completed an exercise and can be things such as a score or a list of mistakes made.

Concurrent feedback, which is what we normally think of as feedback, is feedback which occurs during the solving of the exercise. It is broken into two sub-categories, one which takes place before a user takes an action and one which takes place after the student has taken an action. This distinction is necessary as anticipatory (pre-action) feedback will most likely be different from reactionary (post action) feedback.

Feedback polarity determines the goal of the feedback, for example, is it intended to prevent or correct an error (Negative Feedback) or is it intended to indicate that the learner is on the right track and to further encourage their actions (Positive Feedback). An example of a negative feedback would be highlighting the latest error the learner made. An example of positive feedback would be displaying to the user the name of the concept they just used.

Specificity of feedback determines the actual content of the feedback. It could be a simple encouragement, the notification of an error, or guidance to the correct course of action. This attribute can fall into three categories: General Statements which have no direct relevance to the exercise in question such as "good job" or "are you sure that's right", Exercise Specific Domain Knowledge which is general domain knowledge directly relevant to the question at hand, and, finally, User Action Specific Information which is information directly relating to the user's individual actions and how they will alter the current state of the exercise. It is important to note that the Specificity of feedback refers to the content of the feedback not to the means of selecting the type of feedback to issue. For example the system may issue positive post action general statement feedback in response to the user taking a correct action.

The attributes of feedback outlined above do not necessarily define all of the possible categories of feedback. It is conceivable that attributes other than Timing, Polarity and Specificity can be used to qualify types of feedback. Additionally even within the categories outlined, there may be some ambiguity, particularly when it comes to Specificity. Fitting feedbacks into hard and fast categories is not always trivial.

In the following sections we will examine a number of useful permutations of feedback, providing examples, commenting on the feedback's impact on the learning process, and determining its suitability for use in ill-defined domains.

## 5.2 Permutations of Feedbacks

### 5.2.1 Positive, Pre-Exercise Feedbacks

#### Positive, Pre-Exercise, General Statements

This is defined as positive statements made to the learner before the learner performs an exercise such as: "you're in the top 10% of your class". These are rather simple in nature and are not related to the domain of the ITS. They can allow the learner to gauge their performance relative to their peers or their past. In general, this is an extremely simple form of feedback that can be implemented in almost all cases, including ill-defined domains. In fact, many systems incorporate this form of feedback without expressly acknowledging that it is a form of feedback such as in online games where the user is presented with information about their current ranking relative to other students.

#### Positive, Pre-Exercise, Exercise Specific Domain Knowledge

Some systems take the above idea of providing relevant information to the learner a bit further and in effect provide directly useful information about the upcoming exercise to the learner. This could be things like pointing out directly relevant examples, (similar to when an instructor reviews important concepts before a mid term) or it could be something as simple as explaining big-O notation in an algorithms course or how to do matrix multiplication in a linear algebra course. This is generally quite easy to implement and like the above General Statement feedback, does not require any sophisticated techniques. Many applications will provide a "tip of the day" during the initial startup which can also be thought of as an example of this feedback.

#### Positive, Pre-Exercise, User Action Specific

This permutation of feedback does not appear to have any merit as the user has not yet taken any sort of action and it would therefore make little sense to attempt to talk about their actions. Perhaps if one had a user model of incredible sophistication it might be possible, but to our knowledge no such model exists.

### 5.2.2 Positive, Post-Exercise Feedbacks

#### Positive, Post-Exercise, General Statements

This combination is similar to Positive, Pre-Exercise, General Statements as seen in Section 5.2.1 with the exception that the feedback is being presented after the learner has completed an exercise. This allows the system to directly comment on the learner's performance in solving the last question or exercise. This could take the form of a statement such as "You did well on that question" or an actual quantitative score on the exercise. This form of feedback does not rely on any techniques more sophisticated than producing a simple score or ranking of users and thus is well suited to ill-defined domains, needing only verifiability. An example of this feedback can be found in almost any educational software in the post-exercise score summary presented to the learner.

#### Positive, Post-Exercise, Exercise Specific Domain Knowledge

Similar to the General Statement feedback described above this feedback is comprised of useful information for the learner presented after an exercise has been completed. Unlike the above feedback, however, we are now able to present information regarding domain principles or domain rules which may be relevant, allowing the system to communicate to the learner what they did right in an exercise specific manner. An example of this would be a statement such as "In questions involving integrals of $e^x$ you perform above average". This method requires the system to have some notion of the domain knowledge required for each question, resulting in it generally being outside of the purview

of ITSs in ill-defined domains.

#### Positive, Post-Exercise, User Action Specific

This is the most specific of the Positive, Post-Exercise techniques. The system comments directly on discrete actions that the learner has taken in the process of solving an exercise. This can take the form of highlighting efficiencies or pointing out the deeper meaning of the action taken by a successful attempt. This method of delivering feedback is the hardest to perform in an ill-defined domain and is generally the province of systems used in well-defined domains. It requires that the domain have the features of formal domain theorems and a lack of overlapping sub problems as stated in Section 4. CodeHunt uses this type of feedback to advise the learner about how to fix any problems in the exercise.

### 5.2.3 Positive, Pre-Action Feedbacks

#### Positive, Pre-Action, General Statements

We have now reached the more difficult forms of feedback as they take place not before or after an exercise but during the exercise. This alone requires that the domain possess one of the traits of being well-defined. In order for there to be pre or post action feedbacks it must be possible to break the problem into discreet tasks such as the steps necessary to solve a mathematic problem or selecting regions of an image. This requirement means the task must not contain overlapping sub problems as described in Section 3.1.5 and must also have a well-defined task structure. Assuming this condition is met, positive pre-action general feedback could perhaps take the form of positive encouraging statements delivered before the user acts however, as of this writing we do not know of any systems which implement this. Ultimately, like all general statement based feedbacks, this is the least specific to the domain and hence offers the learner the least assistance. This is not to say that General Statements cannot provide useful information to the learner, for example a statement like “keep going” can tell the user that the system believes they are

on the correct path in a domain which has verifiability but does not have a decomposable task structure and hence is not well-defined.

#### Positive, Pre-Action, Exercise Specific Domain Knowledge

This type of feedback allows some systems to produce quite useful results, such as in ELM-PE [33] where the user is presented with templates of code to fill in, and a list of related concepts and examples of similar code, all relevant to the exercise at hand. Like other Exercise Specific feedbacks, this method requires the system to know things about the exercise in question. In particular it has to know what concepts need to be understood in order to solve the given exercise so that it can present from those concepts. Unfortunately this requires the domain to not be ill-defined, as it needs formal theories, and like all exercise concurrent feedbacks it is necessary for the solution of the problem to be divisible to discreet actions.

#### Positive, Pre-Action, User Action Specific

This is a particularly problematic permutation of feedback as it requires us not only to provide the feedback before the user acts, but also to correctly predict what action they are going to take. As far as we are aware no Intelligent Tutoring System has attempted this (most likely due to the difficulty of producing a user model which can successfully predict user actions, although it is possible that a sufficiently sophisticated statistical model might be able to predict user's actions with some amount of accuracy).

### 5.2.4 Positive, Post Action Feedbacks

#### Positive, Post Action, General Statements

Like the pre-action based methods described in Section 5.2.3, this method requires the exercise be decomposable into discrete actions to which the system can react. It is made up of general statements about the users' actions such as "good job" or "that was the right choice". Though lacking in exercise specific content these statements can be still useful as was stated in Self's work, "only 40% of tutor student dialog has tutorial content" [27]. Like all

forms of feedback the timing must be chosen carefully as one could reinforce an incorrect action by virtue of being unable to specify what was correct.

#### Positive, Post Action, Exercise Specific Domain Knowledge

This feedback contains domain related knowledge. It is a powerful form of feedback where the system can directly state what would be a useful domain principle to the user such as in LogicTutor (Section 3.3) where after the student has acted, the system can suggest the follow up step based on their choice of logical transformations employed in their proof. It requires some form of domain model so as to select the appropriate information to give to the learner generally necessitating formal rules or theorems and well-defined concepts to be present in the domain.

#### Positive, Post Action, User Action Specific

This is one of the most interesting forms of feedback in which the system reacts to how the user has answered the question. That is to say, it is not just reacting to the fact the user has answered the question, it is reacting to the content of their answer. This goes beyond reacting to an improvement or worsening of the state of the exercise and speaks to the actual domain and educational content of the feedback given. An example of this feedback would be providing additional background information as to why the correct choice was correct. This feedback requires the domain be some what well-defined, requiring formal theorems, well-defined concepts, and a well-defined task structure. These features are necessary as this feedback does not just state if the user was correct but why they were correct.

### 5.2.5 Negative, Pre-Exercise Feedbacks

#### Negative, Pre-Exercise, General Statements

As discussed earlier while positive feedback is feedback intended to cause the user to continue a correct course of action negative feedback (the much more common variant) is used to correct the user's actions such as notification of an error, or information which may prevent a future error. Negative pre-exercise

general statements feedback is simply the system making general statements not with the intention of encouraging or discouraging the learner, but with the intention of avoiding future error. For example, a trigonometry ITS might issue feedback like "Remember, the length of the hypotenuse is equal to the square root of the sum of the squares of the other two sides," before commencing an exercise. Incidentally, negative feedback is not negative in tone, in fact it can be quite positive and it is intended to correct the user's mental model of the system.

#### Negative, Pre-Exercise, Exercise Specific Domain Knowledge

Domain knowledge by itself can quite often prevent future mistakes, such as in the case of a statistics application where many potential errors can be prevented by simply stating the probabilities of all possible outcomes must sum to one. This level of specificity requires we have some form of domain model, unlike the one before as we need to know what exact domain concepts are relevant to the upcoming problem. It is possible to hand annotate the exercises with useful domain concepts, but this restricts the possible problems to ones generated from a template.

#### Negative, Pre-Exercise, User Action Specific

Like its positive counter part, this form of feedback requires the ability to predict the learners specific or likely actions. As of this writing the authors know of no system which has such capabilities.

### 5.2.6 Negative, Post-Exercise Feedbacks

#### Negative, Post-Exercise, General Statements

Negative Post-Exercise feedback is a difficult thing to conceptualize as useful but in reality it can be. For example, telling the learner how long their peers took to complete the same exercise, or what the average score was can be helpful (although the latter is only possible in domains with verifiability.) Interestingly, things like a learner's score can be viewed by the learner as

Negative (the learner has a particularly low score relative to his peers) or Positive (the learner is performing better than most of his peers).

#### Negative, Post-Exercise, Exercise Specific Domain Knowledge

This feedback type overlaps with its positive counterpart as, once again, it is restricted to simple statements however now it can contain general exercise specific domain principles. This can take the form of a post-exercise general tip such as "Remember, all statements must end with a semi-colon," when the user has failed to use semi-colons appropriately in a programming ITS. To produce such statements one can either use knowledge of a well-defined domain or hand create appropriate statements for ill-defined domains. Unfortunately, like all of the exercise specific domain knowledge feedback, this requires some way to determine which, if any, domain principles are relevant which generally restricts automatic generation of this feedback to well-defined domains.

#### Negative, Post-Exercise, User Action Specific

This is a common form of feedback in domains where there are no overlapping sub problems and there is verifiability. In its simplest form this feedback can be an enumeration of the mistakes made by the learner examples of which can be found in almost all e-learning software. In a more robust form, in domains with formal theorems or an artificial set of rules, it can take the form of a direct statement. This statement can be used to inform the learner which domain principles were violated and what they can do to avoid the same mistakes in the future.

### 5.2.7 Negative, Pre-Action Feedbacks

#### Negative, Pre-Action, General Statements

This feedback contains only general statements intended to prevent the user from making a mistake. These could take the form of stating general domain principles that the system believes the user is unaware of, or simple messages about upcoming difficulty. Depending on the amount of domain knowledge contained within the message it may not be necessary to have a well-defined

domain to provide this form of feedback. For example, in an ill-defined domain with a well-defined problem structure and a step by step exercise process, it would be possible to record the success rates of students on various steps allowing the system to generate feedback for upcoming difficulty.

#### Negative, Pre-Action, Exercise Specific Domain Knowledge

With this form of feedback the system can express directly relevant domain concepts to the learner before they make a mistake. This could be something such as restating the core principles of the domain to the user, or domain rules directly relevant to the problem. This feedback is mostly restricted to domains in which the system has a notion of what would be relevant information to the learner.

#### Negative, Pre-Action, User Action Specific

This is the most difficult form of feedback to provide as it is extremely difficult to anticipate the learner's forthcoming action as well as what the next correct action should be. This requires a domain which has formal theorems, dividable subtasks, and a lack of an open field task structure. As of writing we do not know of any ITS that uses this form of feedback. That said it may be possible to achieve this by forming a statistical model of reasonable user actions in order to guard against common mistakes.

### 5.2.8 Negative, Post-Action Feedbacks

#### Negative, Post-Action, General Statements

Negative Post-Action feedbacks are the most commonly associated with the name "feedback". They are issued in reaction to the learner taking an incorrect action. They seek to notify the learner of a mistake they have made, and prompt them to correct it. When used with General Statements it is a simple notification that the student has made an error. For example, if the user makes a mistake the system can state "You should examine your last action", prompting the user to contemplate their last act. These can be offered in any domain which has verifiability and the task is one comprised of discreet steps.

#### Negative, Post-Action, Exercise Specific Domain Knowledge

This is a more specific version of the above with only the additional requirement that the system be able to reason about the domain concepts directly related to the given question. Much like the exercise specific methods, this can be done by either having a domain model which can reason about the questions or by direct annotation of the question manually. Such a feedback can be seen in ELM-PE, where the system will indicate if the learner's answer will produce the an incorrect output and then checks to see if this error is part of a list of common errors to resolve.

#### Negative, Post-Action, User Action Specific

This is the most effective feedback variety which simulates the actions of an effective human tutor. It is a corrective statement directly highlighting the user's error or providing the learner with suggestions about how to resolve an error. Unfortunately it requires the domain be well-defined or, at the very least, have the properties of formal theories, verifiability, a lack of overlapping sub problems, non open textured concepts, and a suitable task structure. It is also the most common form of feedback for ITSs in well-defined domains. An example of this can be found in LogicTutor (Section 3.3) where after the student has acted erroneously the system directly indicates the location of the mistake.

## 5.3 Feedback selection in Ill-defined domains

Now that we have enumerated the possible feedback combinations we will now examine the methods with which a system can select a suitable feedback for a learner. In particular we will be examining the efforts to provide feedback in ill-defined domains. We will first examine and enumerate the approaches outlined in Fournier-Viger et al. [10]. We will then examine the coding ITS CodeHunt [25]. And later on, in Section 7.1, we will propose a new method of selecting feedback based on Reinforcement Learning.

### 5.3.1 Existing methods of feedback selection in Ill-defined domains

There are many ways of creating an ITS capable of providing feedback in ill-defined domains. Some examples include forcing structure onto them, creating a partial task model, using existing case studies and using existing evaluation metrics. Over this next section we will touch upon each of these techniques, exploring their advantages and limitations.

When forcing structure onto a domain, one can create consistent but not comprehensive rules for the domain, such as in SlideTutor [9], described in Section 4.1.3. The authors produced a domain model which, while not comprehensive, is sufficient for guiding students to replicate the action of an expert. This is done by using the model to provide positive and negative pre-action exercise specific feedback. A related method is to create constraints on what comprises a valid action such as avoiding mechanical singularities in the CANADARM tutor [10]. This enables it to provide Negative Post-Action User Action Specific feedback in cases where the constraints had been violated. These techniques are suited to many problems, but they cannot replicate the results of a similar system in a well-defined domain. The construction of such rules can also be both time consuming and error prone.

Creating a partial task model is a similar idea, but, instead of forcing a model onto an entire domain, one discards the sections of the domain which are ill-defined and focuses on a well-defined subset of the domain. In domains which support creating a partial task model this can be a powerful technique enabling the use of conventional ITS feedback selection techniques. Unfortunately many domains lack a well-defined segment in which to use this technique for example, writing.

Making use of pre-existing case studies allows a system to apply structure to judgment based domains, such as in PETE [12], where the authors have curated a set of case studies from which students can learn and explore. This technique has some value as it allows us to impose verifiability on a domain which would otherwise not have it. For example, this technique could enable

the system to inform the learner they have missed an ethical concept so they can attempt to replicate the judgement exhibited by experts in the field. In effect this is a form of Negative Post-Action Feedback.

Finally we come to using existing evaluation metrics. This can be doing something like key-word analysis or making use of a writing quality metric. It enables us to impose some form of verifiability on domains where such metrics exist, allowing the use of at least general feedbacks.

### 5.3.2 Collaborative methods of feedback selection

Collaborative methods are an effective alternative to the methods outlined in the prior section. Instead of imposing structure or attempting to acquire structure from case studies, these methods use the most proficient students as a template for those less proficient. An example of this can be found in CodeHunt [25], which is described in depth in Section 4.1.1. Students using CodeHunt are tasked with writing a small computer program which is required to pass some tests. If their program fails these tests their solution is compared with the history of other learners. From this the system can recommend a next course of action. The system looks for learners who were in a similar situation and later resolved it. Once such a learner is found, the successful resolution is provided to the learner as a User Action Specific feedback. This method is interesting and worth pursuing in domains which can support it i.e. ones in which novelty is not desired. It is suited to both problem solving domains and analytical domains.

# Chapter 6

# Shufti

Shufti is an Intelligent Tutoring System that focuses on teaching mammography - the diagnosis of mammograms or x-rays of the breast. An example of a mammogram can be found in Figure 6.1.

Shufti is a Ruby on Rails application using PostgreSQL as its store of data. It is made up of 14861 lines of Ruby and 1822 lines of Javascript. It makes use of competitive features, game design, and automatic custom feedback to aid learners and to encourage them to practice their skills with the system.

Learners are presented with exercises which they complete as a form of practice for mammography. There are four kinds of exercises: Presence exercises, Dual Image Heat Grid exercises, Dual Image Grid exercises, and Dual Image Polygon exercises. Exercises are ranked by difficulty with Presence exercises being the easiest and Dual Image Polygon exercises being the most difficult. Users are limited to only one difficulty or "level" at a time and are promoted based on their performance. A total of 1883 mammograms drawn from the Digital Database for Screening Mammography [13] were used for these exercises.

We will now turn to Shufti's design and how this relates to Shufti's pedagogical goals and use of gamification to accomplish those goals.

## 6.1 User Summary Screen

Shufti's User Summary Screen (Figure 6.2) presents users with a variety of information including a graph showing how their score has progressed, a list of

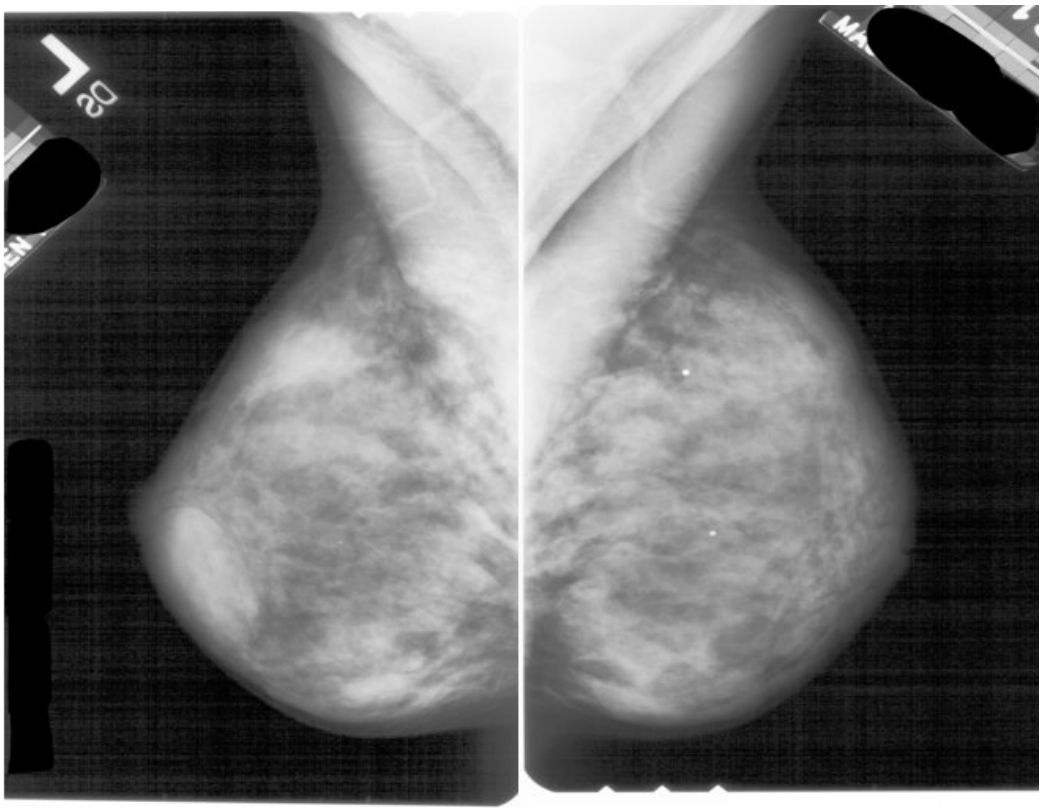

Figure 6.1: An example mammogram.

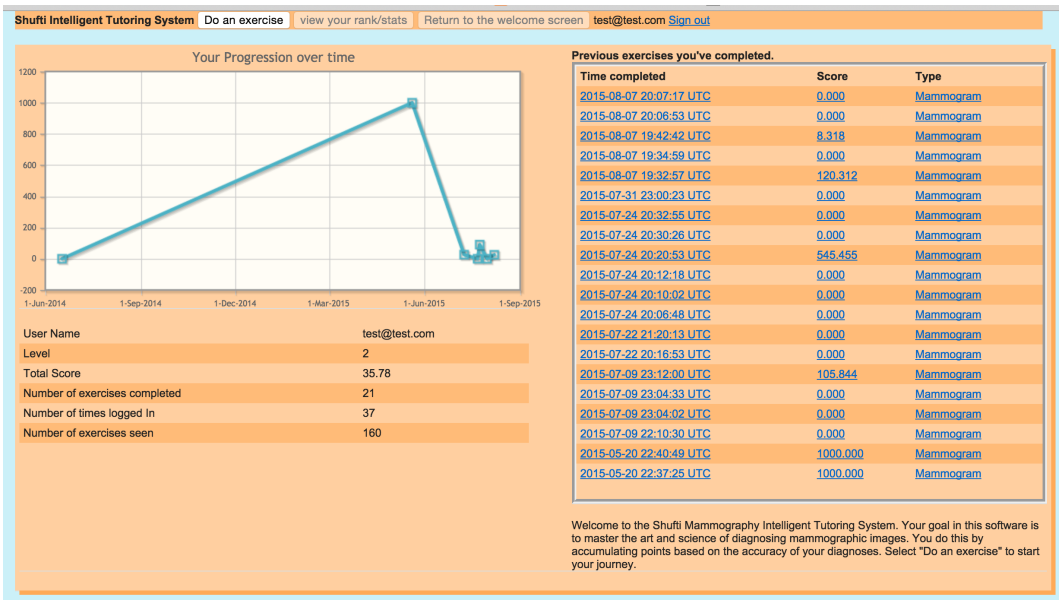


Figure 6.2: The User Summary and Home Screen for Shufti

their past exercises, and scores with links to details about each specific exercise and, finally, a set of statistics summarizing their level, usage, overall score and number of exercises completed.

The purpose of the summary screen is to provide the learner with a concise summary of their activities and to serve as a jump-off point for all the other functions in Shufti such as starting an exercise or reviewing past exercises.

## 6.2 Exercises

The main component of Shufti is the mammography exercises. These exercises task the learner with making diagnoses of varying difficulty and goals. When a learner is done with an exercise they submit it for evaluation. They receive a score and a summary of errors and correct diagnoses. They have the ability to cancel an exercise although this results in a score of Zero.

Learners are given various tools such as zoomed views and (on exercises harder than Presence exercises) automatic feedback to help them complete the exercises.

Once they have completed their exercise they can see the correct answer, how their peers preformed, a replay of their mouse movements and the com-

ments of their peers on the exercise.

We will now examine the unique features of each type of exercise and explain how a learner would experience them.

### 6.2.1 Presence Exercises

Presence exercises are the simplest ones. In them the learner determines if a lesion is present within the provided mammograms. They give a simple "yes" or "no" answer to each exercise. This is similar to a coarse attempt at determining if there's anything unusual present in the breast tissue at all. The exercises are of limited difficulty showing large lesions when present. The purpose of these exercise is to acquaint the learner with mammograms and teach them to identify fairly obvious lesions (the minimum size of lesions in Presence exercises is 32 pixels by 32 pixels). Learners in this exercise are scored as a simple pass fail with a correct answer giving them a score of 1000 and a incorrect answer a score of 0, this range was chosen to match the range of scores in the other exercises.

### 6.2.2 Dual Image Heat Grid Exercises

Dual Image Heat Grid Exercises task the learner with directly indicating where they believe lesions exist, if any, in a given mammogram. This is to have the learner practice locating any cancerous lesions within the breast instead of just the fact that the are present. They do this by selecting squares in an 8 by 8 grid which has been superimposed over each breast in the mammogram. Each square in the grid is an average of 100 pixels by 66 pixels. This proved to be ergonomically suitable. When a user is satisfied with their answer they submit their exercise for evaluation.

While the learner is answering an exercise they are given "Hot" and "Cold" feedback (similar to the game in which one person guides another to an object or location by continuously stating if they have moved closer to (warmer) or further from (colder) the object). In Shufti this takes the form of the thermometer displayed in the right region of Figure 6.7. The thermometer changes to yellow if the user hovers over a square with a immediately adjacent

$$F_1 = 2\frac{precision \cdot recall}{precision + recall}$$

Figure 6.3: $F_1$ score

to a square containing a lesion and it turns red if the user hovers directly over a square which contains a lesion. At all other times the thermometer remains blue. In order to prevent the learner from abusing the thermometer to quickly produce a correct answer, the thermometer only changes yellow or red if a learner has hovered over a particular square for at least 30 seconds (assuming that square contains or is adjacent to a lesion).

Once a learner submits their answer Shufti assess it and gives them a score. This score is determined by multiplying the $F_1$ score [15] by 1000 in order to give a score out of 1000. $F_1$ score, a common statistical measure used in document retrieval, is the harmonic mean of a answer's Recall (the proportion of all of the lesion-containing squares in the mammograph correctly selected by the user) and precision (the proportion of the squares selected by the user which correctly contained a lesion). For example consider the exercise answer visible in Figure 6.9. In it the yellow boxes are the answer, the green areas are where the user selected correctly and the red areas are where the user selected incorrectly. The precision is defined as the number of correctly selected squares divided by the total number of selected squares, in this case 5 squares are selected correctly, and a total of 9 squares were selected giving a result of 5/9 or $0.\overline{5}$ for precision. The recall is defined as the number of correctly selected squares divided by the number of squares in the answer, in this exercise 5 squares were selected correctly and the entire answer is 8 squares large producing a recall of 5/8 or 0.625. The formal definition of $F_1$ score can be seen in Figure 6.3 and by using the numbers from the above example we get a $F_1$ score of 0.59. The only exception to this is when there are no lesions in the mammogram in which case the user's score starts at 1,000 and they are penalized by 200 points for each erroneous selection (with the minimum score being zero).

An example of the process of answering a Dual Image Heat Grid exercise

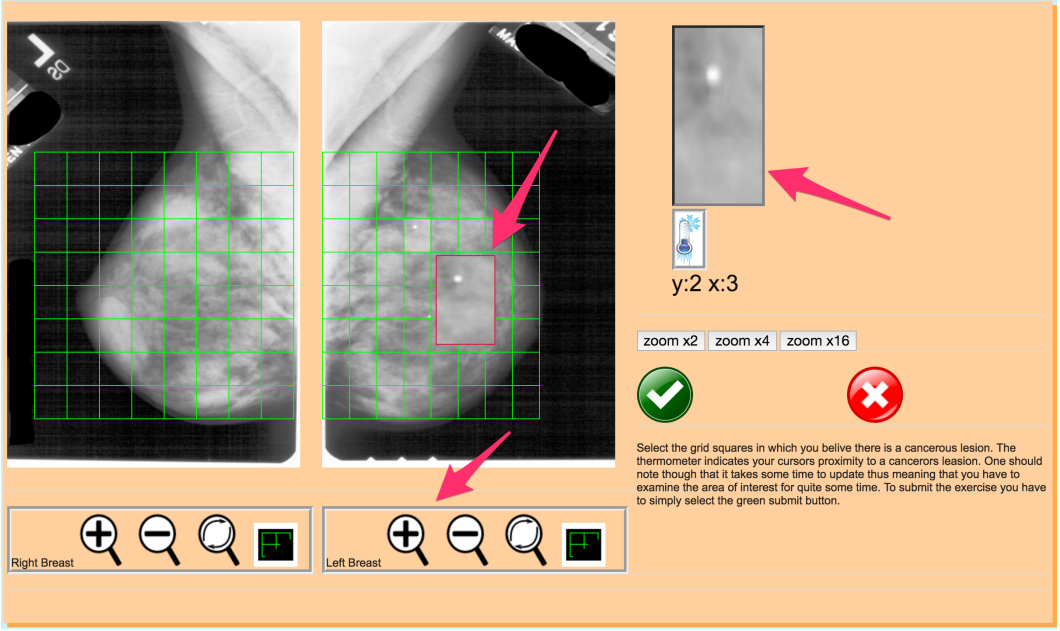


Figure 6.4: A view of a exercise with the zoom controls present.

can be seen in Figures 6.4, 6.5, 6.6, 6.7, 6.8, and 6.9. In Figure 6.4 we see the various zoom tools available: a mouse following zoom that displays the region immediately around the mouse, a square zoom which shows a zoomed view of the last grid square passed over, and the whole image zoom which can be used to magnify the entire image. In Figure 6.5 we can see a cold feedback being issued indicating that the mouse cursor has been away from the location of the lesion for some time. Figure 6.6 we can see a warm feedback that indicates that there is a lesion near the users cursor. In Figure 6.7 we see a hot feedback which indicates that there is a lesion under the students cursor, this information is only given if they hover over the square for some time. Next, in Figure 6.8 we can see a possible solution to the exercise. In Figure 6.9 we can see the results of submitting the answer in Figure 6.8. In Figure 6.9 the yellow outlined boxes represent the correct answer, the green filled boxes represent where the student has selected correctly, and the red filled boxes represent where the student selected incorrectly. Also in Figure 6.9 in the overlain red box we can see the replay controls, which enable the learner to view their own mouse movements over the grid.

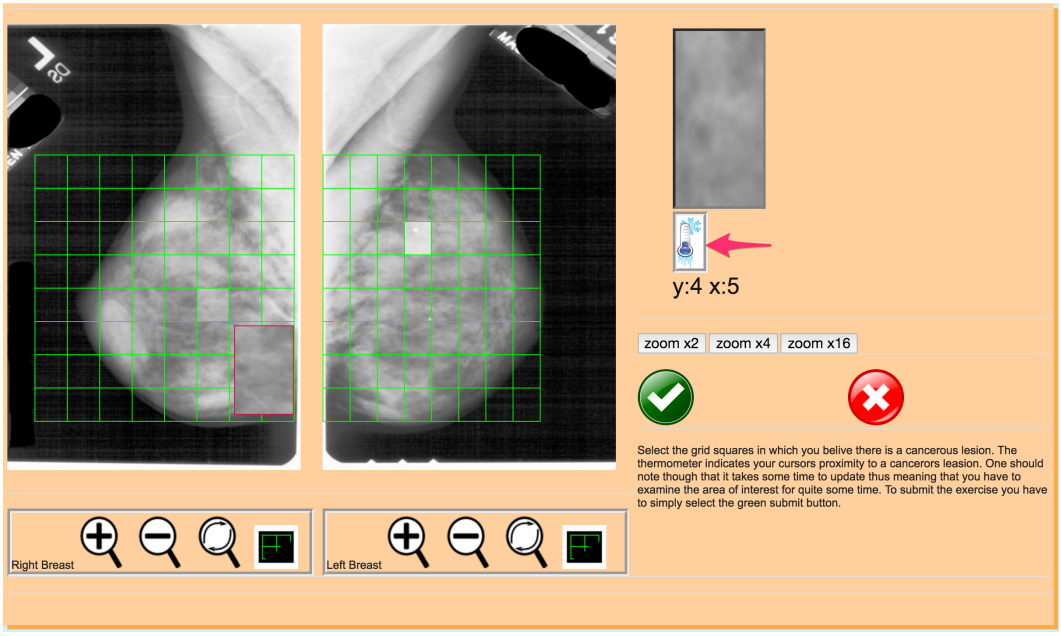


Figure 6.5: A cold feedback has been issued as can be seen from the blue or cold thermometer.

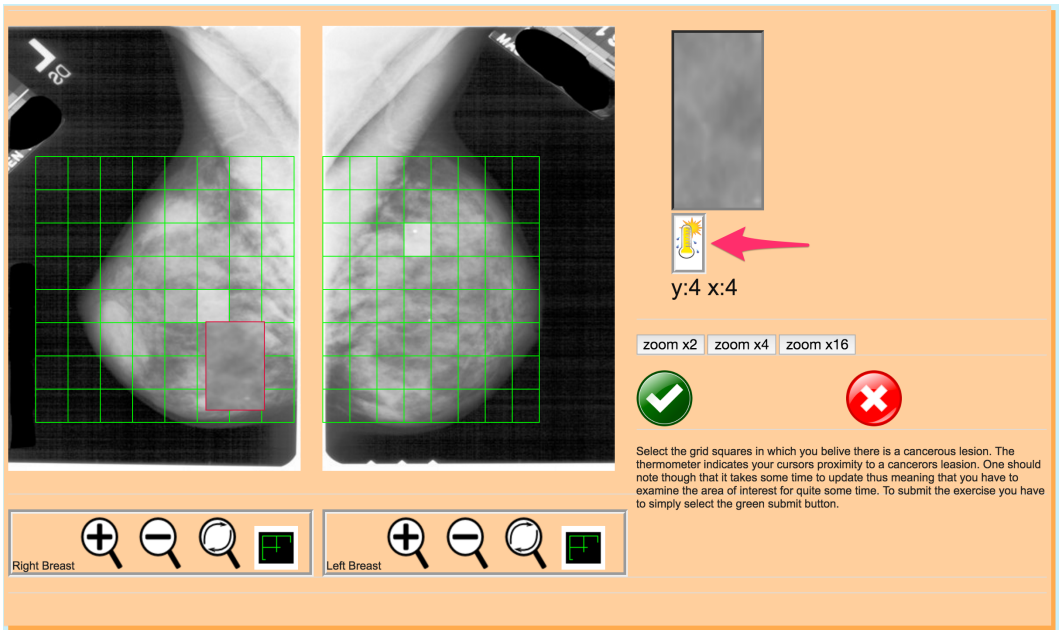


Figure 6.6: A warm feedback has been issued as can be seen from the yellow thermometer.

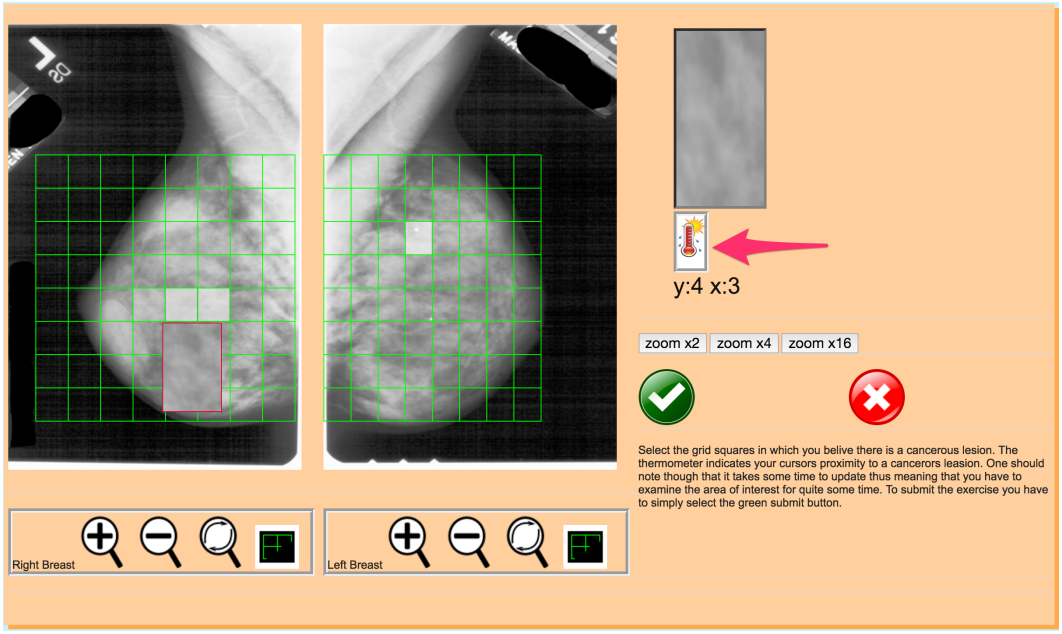


Figure 6.7: A hot, or red, thermometer means our mouse of hovering over a lesion.

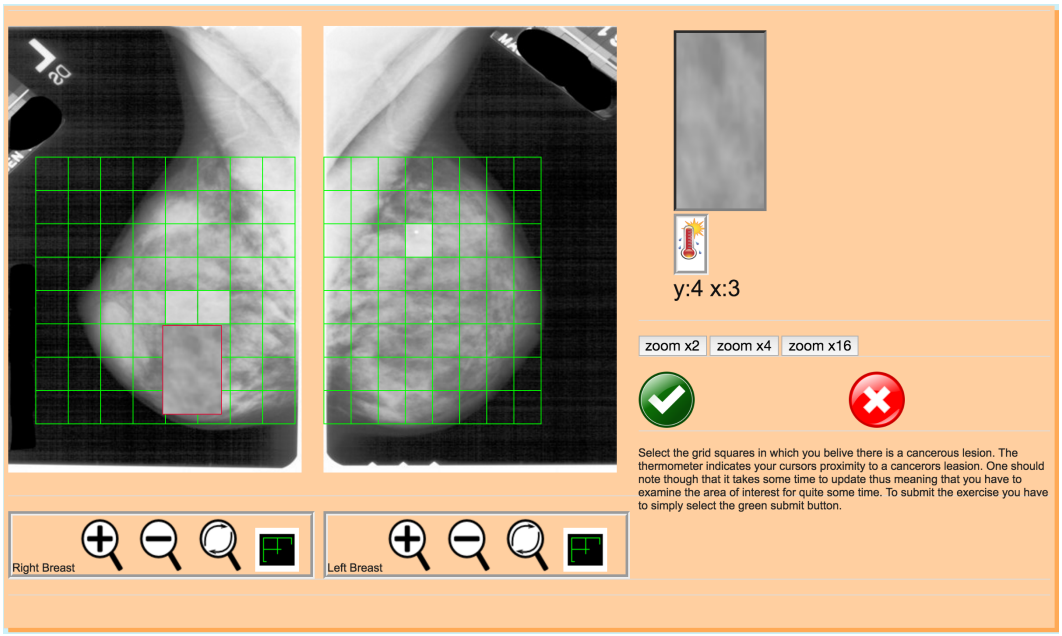


Figure 6.8: An example selection of where the student thinks the lesion is.

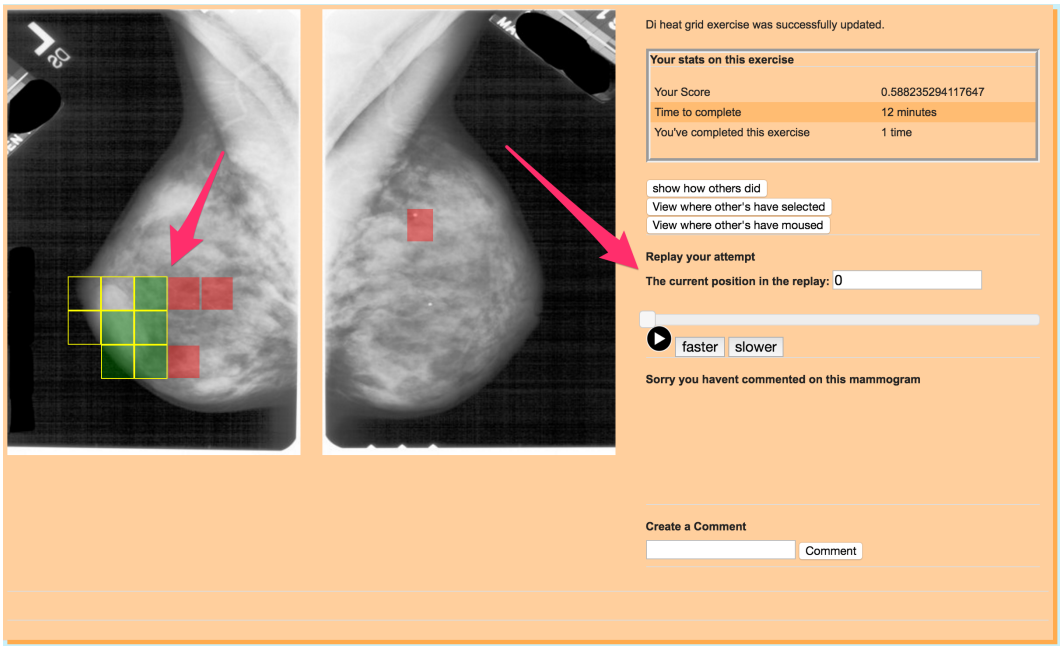


Figure 6.9: A view of the after exercise .

### 6.2.3 Dual Image Grid Exercises

Dual Image Grid Questions are substantially harder than Dual Image Heat Grid questions because the hot and cold feedback is no longer offered to the learner. All other features are the same including the user interface and score computation.

### 6.2.4 Dual Image Polygon Exercises

The final and most difficult exercise variety is the Dual Image Polygon exercise. In Dual Image Polygon exercises, the learners are tasked with specifying the exact shape of a suspected legion by defining a polygon overlaying the image. This approximates how a radiologist indicates the extent of a lesion using a marker circling it's extent, and better matches the polyignal lesion areas in our answer set. To create this polygon they select the polygons vertices one by one on top of the mammogram. These vertices can be moved or deleted in order to further refine the shape of the lesion. An example of this process can be seen in Figure 6.10.

Once the learner believes they have adequately defined the lesion or lesions

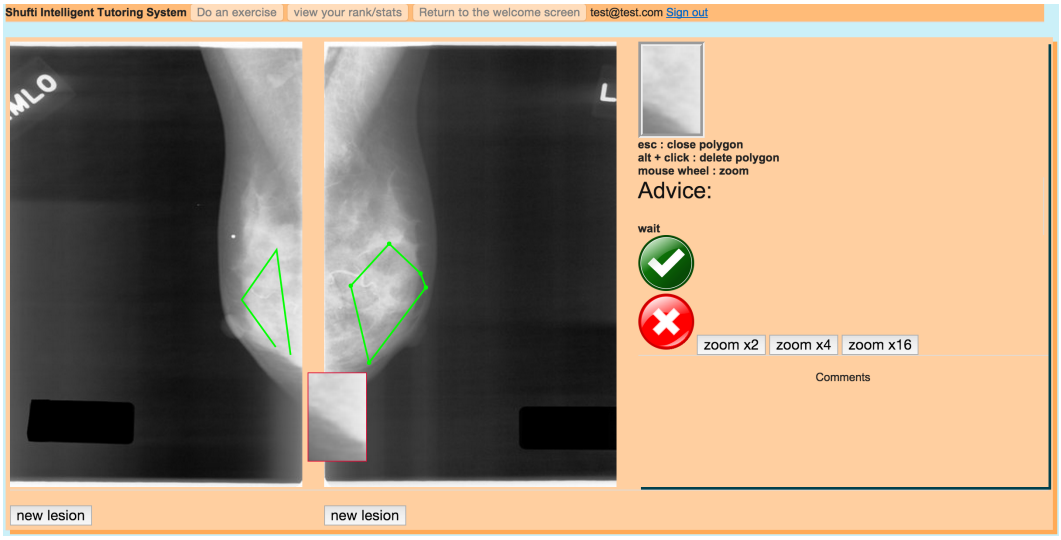


Figure 6.10: Dual Image Polygon Exercises's User Interface

they submit their exercise and are given a score. This score is also based on the $F_1$ score, as was described in Section 6.2.2, but how the Precision and Recall are computed has changed. Recall is now computed as the size of intersection between the selected areas and the correct areas divided by the total correct area's size in terms of pixels. Precision is now also in terms of pixels and is computed as the size of the intersection of the selected areas with the correct areas divided by the total selected area. This results in both the minimizing incorrect areas, while also not missing any lesion area being valued.

### 6.2.5 Promotion

After exploring each of the possible levels and exercise we can now talk about how users move between each one. In Shufti students are assessed and given any appropriate promotions after an instructor defined number of exercises. They are assessed to see if they have exceeded an instructor determined performance level since their last evaluation. These parameters are set though a configuration page available to the instructor. This design was chosen for three reasons: First it prevents students from becoming trapped at a level by an early performance as in the case where the students entire history is considered. Second it prevents a single lucky exercise from promoting a student who shouldn't be, as is possible when only the students most recent

performance is considered. Lastly by enabling a courses instructor to tune the parameters used for promotion the rate at which leaners advance in difficulty level is determined by instructors opinion on their students abilities and time constraints.

## 6.3 Pedagogical Goals and Gamification

Shufti's Pedagogical goals are to improve the breadth and depth of the knowledge of medical imaging instruction. It is desirable for the learners to see and analyze as many cases as possible. It is also desirable for them to gain as much value out of each viewing as possible. To accomplish this, Shufti makes use of modern gamification techniques, to have the learners view more cases than they normally would, while also causing them to focus on their own performance thus leading to deeper practice [24]. Gamification as described in Yee's Pedagogical Gamification [35] is a set of design principles used in the creation of a set of challenges, or in education's case exercises. These design principles are: Displaying Progress, Maximizing Competition, Careful Difficulty Calibration, Providing Diversions, and Employing Narrative Elements. We will now examine each of these principles and how it relates to Shufti's design.

### 6.3.1 Displaying Progress

The display of a player progress is an important part of the game experience. By showing how far a player has come they are encouraged and will continue to play. In a serious game learners continue to practice encouraged by their growing proficiency [11]. In Shufti, progress is displayed to the learner through three means. First the learner's score is prominently displayed at the completion of an exercises. The learner's score is also tracked by a summary graph on the summary page (Figure 6.2) enabling them to view their overall progress. Second as they improve their skills they can ascend to a new level where they are presented with more difficult exercises further indicating their progress. Finally the learner has a chance of seeing the same exercise again, when this happens they are notified at the en of the exercise, enabling them to see their

own progress directly.

### 6.3.2 Maximizing Competition

Humans are naturally competitive, valuing the ability to compare oneself with peers. To succeed against others can be a powerful drive. To encourage this, Shufti optionally provides learners with an anonymized ranking where they can compare themselves with their peers. This is similar to the leader boards present in many popular online games. It also allows learners to see how their peers answered exercises by overlaying colours on the mammogram indicating the frequency that areas were selected as containing malignant lesions, enabling a more focused comparison.

### 6.3.3 Careful Difficulty Calibration

Challenge can be a powerful motivator, it promotes engagement and overcoming it is very motivating. But too much challenge can make a task appear impossible and be discouraging. Likewise too little challenge results in a task feeling trivial or boring. To balance these two requirements Shufti calibrates its challenge through the use of the levels described in the beginning of this chapter. In Shufti, students are first presented with the lowest level of difficulty, Presence Exercises. Once students have demonstrated mastery they ascend to the next level. This process of mastery and ascension continues until they reach the most difficult exercise type Polygon Exercises. This enables Shufti to adapt to the learner and their proficiency providing both a constant source of challenge and a display of the user's progress.

### 6.3.4 Providing Diversions

It is also important to break up the learning experience, to reset the learner's attention. This can be done though diversions not directly related to the core learning task. In Shufti's case there are many things that learners can do other than directly answering exercises. They have the ability to comment on the exercises they have seen and to see their peers comments. They can see

replays of their mouse movements over past exercises to review their actions. They can also annotate the exercises so when they solve them again they can see their past opinions and considerations.

### 6.3.5 Narrative Elements

Many games also provide narrative elements to further entice the learner. Unfortunately the analysis of medical images does not lend itself to this. Thankfully though it is not necessary for a game to have all of these principles to be engaging [35]. A good example of this is chess which is quite engaging but has no story at all.

Unfortunately at this time we can only speculate about the predicted effects of gamification on leaner engagement in Shufti. This is due to our lack of access to classes necessary to assess it. If possible in the future it would be desirable to preform an experiment on two mammography classes: one given access to Shufti with all of the gamification elements and a control class where the gamification elements of Shufti are not present. We could then compare the rate of usage of the system between the two classes to determine if there are any effects on learner engagement.

## 6.4 Mammography as an Ill-Defined Domain

Medical Imaging and by extension mammography is an ill-defined domain as it lacks some of the attributes of a well-defined domain as is argued by Crowley et al. in SlideTutor: A model-tracing Intelligent Tutoring System for teaching microscopic diagnosis [9]. Mammography lacks formal theories (Section 3.1.2) as there is no theory of interpreting lesions like there would be in physics. This lack of formal theorems means that the interpretation of mammograms, including the location of lesions within them, is based on the practitioners experience and judgment incorporating many non formal sources of information and lacking a step by step process [26], thus mammography does not have a well defined task structure (Section 3.1.3). Another argument for why mammography does not have a well defined task structure is that

as far as we are aware there is no way to automatically generate realistic exercises within the domain. It also does not have clearly defined concepts (Section 3.1.4), for example the line between a cancerous melanoma and a benign one is not clear. However, medical imaging does have some properties which make it possible to sill provide feedback namely verifiability (Section 3.1.1) and discreet sub problems (3.1.5). Medical imaging is verifiable when there is human expert annotations available for use as a gold standard. Also diagnosis of a medical image is comprised of independent of sub tasks, such as each exercise is self contained. Due to these attributes it is possible to still provide feedback to the learner.

# Chapter 7

# Feedback Selection in Shufti

In this chapter we will examine the reinforcement learning based feedback selection method used in Shufti to select feedback for learners. This includes what reinforcement learning is, how it is employed in Shufti, and a selection of algorithms evaluated for use. Additionally a rejected clustering based method is examined.

## 7.1 Reinforcement Learning For Feedback Selection

As mammography is an ill-defined domain, discussed in Section 6.4, it is necessary to develop new methods with which to select feedback for learners. There are many existing tactics with which to provide adaptive feedback in both well and ill-defined domains. For example, in mathematics, a well-defined domain, one can use a computer algebra system to detect errors in the students actions and then indicate any errors found to the student. This results in a form of Negative Post-Action User Action Specific feedback. For ill-defined domains more unconventional techniques can be used. In Perelman and Grossman's Code Hunt [25], detailed in Section 4.1.1, how prior students resolved errors is used as the basis for corrective feedback for students. Other existing methods of providing feedback to leaners in ill-defined domains are discussed in Section 5.3.

We propose a new method of providing learner specific adaptive feedback, which is in use in Shufti. This new method makes use of reinforcement learning

algorithms [29] to model individual students to allow the system to determine the most effective feedback strategy for a learner, even in the case that no feedback at all is the best choice.

Reinforcement Learning is a branch of artificial intelligence which learns how to solve non stationary problems in real time. It learns and adapts to the problem even as the problem changes. These attributes are important as learner's feedback needs can shift over time and the system must be able to react to changes and novel situations. The high level explanation of Reinforcement Learning is thus: given an environment which provides a reward signal and a state signal, control an agent/actor within that environment such that over time the reward signal is maximized.

To characterize providing feedback to the learner in such an environment, we did the following. First we chose a state representation derived from the user's score if they submitted the exercise immediately. For the reward it is also produced from the same score as the state signal, in this case it is simply the score that the user would receive.

Once given the state and reward signals the agent chooses from a list of possible feedbacks to issue to the learner. The list is a set of General Statements and, advice for the learner, labeled with a polarity, either positive or negative. Positive feedbacks are reinforcing in nature, advising the learner to continue with their current course of action. Negative feedbacks are correcting in nature, and attempt to dissuade an incorrect course of action.

When the agent has issued a feedback it then updates its estimation of what the feedback will do in the previously visited state situation. These estimations are know as $Q$ values.

Each learner has their own agent which, over time, learns their preferences when it comes to feedback. The reinforcement learning algorithm learns which feedbacks are effective in improving their performance and which ones make the learner act incorrectly. This results in the system eventually issuing only the feedbacks the learner likes and which are effective in helping them learn.

In the following sections we will first examine some select Reinforcement Learning Algorithms, Then we will explain our choices when it comes to the

characterization of providing feedback.

### 7.1.1 Reinforcement Learning algorithms

For this project we evaluated the following algorithms: SARSA [29], Q-Learning [32], and SARSA Lambda [29]. These algorithms all employ Temporal Difference (TD) learning methods and were chosen due to their ability to correctly attribute reward to actions and the ability to learn while acting. This is in contrast to Monte Carlo methods which learn in a single batch [29].

SARSA (State Action Reward State Action) is the simplest of the algorithms chosen and perhaps the simplest TD learning based method. A SARSA agent works by maintaining a set of state actions and values $Q(s, a)$ which are its estimates of the values of the states. To take an action in state s it selects from the possible Q-Values for that state. This selection is $e$-greedy which means the agent selects the action a with the highest Q-Value, but $e$ of the time it acts randomly so as to explore. The agent then takes action $a$ and observes the next state $s'$ and the reward $r$. It then updates its Q-values using the following update rule $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$, where $\alpha$ is the learning rate $\gamma$ is the discount rate for future rewards and $a'$ is the action that it will take next in $s'$. Over time its Q-Values will converge to their true value under the $e$-greedy policy, this is known as an on-policy update rule.

Q-Learning on the other hand uses an off-policy method. As stated previously, Q-Learning is a TD method in that it learns as it acts. It works much as SARSA does but with one difference - instead of SARSA's update rule it uses the following: $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma max'_a Q(s', a') - Q(s, a)]$, where $\alpha$ is the learning rate and $\gamma$ is the reward discount. Unlike SARSA however, Q-Learning uses the maximum possible Q-value in its update rule, not the one specified by the policy. Because Q-Learning chooses the max Q-Value to use in its update rule instead of the one specified by its policy makes it an off-policy method.

Despite the advantages of SARSA and Q-Learning over other RL methods, both methods do have a downside. In both of these algorithms the update rule only alters the last Q-Value of the last state and action, which means that it

takes some time for the Q-Values to reflect rewards received after visiting them. The solution to this problem is to use eligibility traces. Eligibility traces are values associated with a state action pair that result in past pairs receiving discounted changes to their Q-Values based on currently received rewards.

The RL algorithm we use to incorporate eligibility traces is SARSA-Lambda, which uses a value $\lambda$ to specify the discounted rate at which current rewards are applied to past Q-Values. Its update rule is closer to an update process and works as follows. For the last state and action $s$ and $a$ the eligibility trace $e(s, a)$ is incremented by one and then the current steps delta is computed as $\delta \leftarrow r + \gamma * Q(s', a') - Q(s, a)$ where $s'$ and $a'$ are the next state and the on-policy action that will be taken. Next, every Q-Value is updated using the rule $Q(a, s) \leftarrow Q(s, a) + \alpha e(s, a)\delta$, and then, finally, all the eligibility traces are updated by $e(s, a) \leftarrow \gamma\lambda e(s, a)$. This process results in each past state action pair being given a slowly descending level of credit for currently received rewards, improving the learning rate considerably.

We will compare these methods in Section 8.2.

### 7.1.2 Characterizing Medical Image Analysis as a Reinforcement Learning Problem

Now that we have described the algorithms that are used in the feedback selection we can now be more specific in how we characterized the selection of feedback as an RL problem. How a problem is posed to an RL system, the states, rewards, and possible actions, is just as important as the algorithm used to solve the problem. An incorrectly posed problem can have the RL algorithm solve the wrong problem or learn in an inefficient manner thus it is extremely important to properly pose the problem that should be solved.

The current score, as though the learner had submitted the exercise, is computed. It is then discretized into a state signal with three possible values and is used as a state signal for the Reinforcement Learning agent. The discretized score is placed in the ranges of greater that 0.5, less than 0.5 but greater than 0.25 and less than 0.25. These ranges were chosen to efficiently represent the difficulty of the mammogram diagnosis and to make the learning

rate of the agent as rapid as possible. The reward signal of the agent is again the learner's score but it is not discretized. This reward signal was chosen such that the system would learn the most effective feedbacks as quickly as possible by observing its feedbacks effects during exercises.

The Reinforcement Learning agent is limited to positive feedbacks if the learner has been improving their answer and is restricted to negative feedback when the learner worsened the state of the exercise. This was done to both bootstrap the learning process of the agent (it should not discourage correct actions) and to ensure that the feedback is always of the correct type for the medium term actions of the learner.

The division between negative and positive feedback is important as it prevents the agent from issuing feedback which does not match the current situation for example, issuing encouraging, positive, feedback when the learner has made a mistake. At all times it is possible for the reinforcement learning agent to choose to issue no feedback and, in fact, that is a common choice.

## 7.2 Clustering Students to select feedback

Another method of providing feedback to learners is a much more naïve approach of simply asking them if they liked a given feedback. For example by using a simple prompt after a feedback has been issued. It is then possible to use techniques found in recommender systems to select a feedback for a learner [16] [17]. This is done by clustering the learner's existing feedback preferences on past exercises and placing them with a pear group, then using that knowledge selecting the feedback which was liked the most by the users peers. We did attempt to use this approach and a early version of Shufti, unfortunately there are some issues with such an approach. First the cold start problem will result in many poor feedbacks being issued to the learner. Second there is a very large number of possible exercises so the dataset will likely be sparse. Third using the current approach it applies feedbacks based on the current exercise, not on the current situation, if we instead break up each exercise into various sub states we then worsen the sparsity problem. Fourth and perhaps

most critically prompting a user if they liked the issued feedback is rather disruptive. Ultimately we chose to use the reinforcement learning based approach due to its lack of these issues.

# Chapter 8

# Evaluation

This chapter describes how the feedback system was evaluated. Normally when evaluating a system such as Shufti, a real classroom evaluation is desirable. A real classroom evaluation would involve two mammography classes one with access to the system and one without access, preferably with all the other course materials being equal. We would then compare the performance of the two classes to determine the effect of Shufti. Unfortunately we were unable to obtain access to such a pair of classes as mammography is taught in small groups during a doctors practicum. To still produce a validation we have instead opted to use a simulated class for evaluation. We have created a simulated course with simulated learners, with which to evaluate the correctness of our feedback system. This has been argued for by Self [27] in which he argues, that much as with the design of other complex systems such as aircraft, it is desirable to determine the performance characteristics of a design before using it in a real world context. This simulation enables us to do many things that a classroom evaluation does not. A simulated course evaluation enables us to quickly iterate on our systems design, bugs which could invalidate months of waiting for results are detected in hours. It enables us to experiment with what underlying algorithm provides us with the best performance. Once we have selected an algorithm we can the engage in parameter tuning to find the set of parameters which produce the best result, instead of guessing as we would have to do under a class room evaluation. It also enables us to see the effects of different numbers of feedback types, e.g. can the system preform well

with lots of possible feedbacks to issue or is it limited to lower numbers. Given the advantages and availability of a simulator based evaluation we ultimately chose to create such a system.

## 8.1 Simulator Design

When one does not have access to suitable students, simulating learners is a useful technique to evaluate the performance of an ITS's design. To this end we created a simulator for evaluating Shufti's feedback selection mechanism using a number of "episodes" as follows. Artificial learners were generated and then presented with a series of randomly selected exercises. The learners solved these exercises while the system provided them with feedback. Once the process was complete we evaluated how well the system performed by comparing the average learner's score across a fixed number of exercises, with other episodes using different settings and parameters. It is important to note that we have endeavoured to make this simulator as realistic as possible in particular with regards to the effects of feedback on the learners. Any given feedback type has equal probability of causing a simulated learner to perform worse than they could have or better. Which is to say that each simulated learner has their own randomly generated feedback preferences. Over the next few sections we will examine how an exercise is performed by a learner, how a learner is generated, and finally how a complete episode is performed.

### 8.1.1 Simulated Exercises

There are a number of steps in a simulated exercise. Initially, the learner is presented with a new exercise and the feedback subsystem is given an opportunity to issue a feedback. The learner will then choose which action to take, and will decide if it is happy with the state of its current solution. If so, the learner will submit the exercise for evaluation. If not, the process repeats from the second step, when feedback can be issued. This order was chosen to provide the feedback system with the opportunity to issue a feedback before the simulated learner has acted at all, much as it would in a real world scenario.

When the feedback system chooses to issue a feedback based on the current state of the exercise, the learner reacts to it by temporarily altering the likelihood it will worsen the exercise state rather than improve it. This is done by changing $p_{wrong}$ using the following equation $p_{wrong} \cdot feedbackEffect_n$. Where $p_{wrong}$ is the likelihood of the learner making a mistake, and $feedbackEffect_n$ is the effect that the given feedback has on the learner When $feedbackEffect_n$ is above 1 then the feedback's effect is to worsen the learner's performance on the exercise, if it is lower than 1 then the feedback improves learner's performance on the exercise. It is important to note that the change to $p_{wrong}$ is specific for each feedback type and learner. This is done in order to simulate how real learners react in different ways to different kinds of feedback.

The first step taken by the simulated learner when choosing an action is to decide whether it will take an action that will improve the exercise state or one which will worsen it.This decision is made by comparing its $p_{wrong}$ with a random number in the range 0 to 1. Once the learner has decided, it changes the current answer so as to reflect its choice. The mechanics of improving or worsening the exercise state are specific to the exercise type, and in the case of our simulator we chose to focus on Shufti's dual image grid exercises as described in Section 6.2.3.

To improve the exercise state for a dual image grid question the simulator follows the following algorithm:

1. Gather the yet to be selected correct squares and put them in $S_{answer}$.
2. Gather all the currently selected squares and put them in $S_{attempt}$.
3. Take the intersection of $S_{answer}$ with $S_{attempt}$ and place it in $S_{correct}$. This will produce the set of currently selected squares that are also part of the correct answer.
4. Remove any member of $S_{correct}$ from $S_{attempt}$ and store the result in $S_{wrong}$.
5. Gather all squares that are neighbours of the currently selected squares and are also part of the answer and place them in $S_{neighbours}$.

6. Place all the un selected non neighbour squares which are part of the answer in $S_{possible}$.

7. Now randomly designate a square to improve from $S_{neighbours}$, $S_{wrong}$, $S_{possible}$, With the weights of 1, 0.6 and 0.5 respectively, or in other words, a square in $S_{neighbours}$ is twice as likely to be selected as any give square in $S_{possible}$.

8. Finally take the previously designated square and if it is in $S_{neighbours}$ or $S_{possible}$ place it in the current answer by selecting it. If it is in $S_{wrong}$, remove it from the current answer by deselecting it.

This will result in the exercise being improved in a believable manner, with adjacent correct answers more likely to be added than wrong answers removed and adding distant correct squares as the least likely activity.

To worsen the exercise state the learner follows a similar process:

1. Gather the yet to be selected correct squares and put them in $S_{answer}$.

2. Gather all the currently selected squares and put them in $S_{attempt}$.

3. Take the intersection of $S_{answer}$ with $S_{attempt}$ and place it in $S_{correct}$. This will produce the set of currently selected squares that are also part of the correct answer.

4. Remove any member of $S_{correct}$ from $S_{attempt}$ and store the result in $S_{wrong}$.

5. Gather all squares that are neighbours of the currently selected squares and are also not part of the answer and place them in $S_{neighbours}$.

6. Place all the un selected non neighbour squares which are not part of the answer in $S_{possible}$.

7. Now randomly designate a square to worsen from $S_{neighbours}$, $S_{correct}$, $S_{possible}$, With the weights of 1, 0.625 and 0.375 respectively, or in other words, a square in $S_{neighbours}$ is twice as likely to be selected as any give square in $S_{possible}$.

8. Finally take the previously designated square and if it is in $S_{neighbours}$ or $S_{possible}$ place it in the current answer by selecting it. If it is in $S_{correct}$ remove it from the current answer by un selecting it.

After performing this process the answer will either contain one more incorrect square or one less correct square thus worsening the state. As with the improvement scenario, the weights were chosen in order to represent learner's tendency to select squares adjacent to the existing selection.

Once the simulated learner has acted the simulated learner decides if it is going to submit the exercise or continue. It does this by determining if the following is true $100 <= timeWeight * numberOfSteps + scoreWeight * score$ where $timeWeight$ represents the impatience of the learner, $numberOfSteps$ is the number of times the exercises answering process has taken so far, $scoreWeight$ represents the learner's ability to estimate if the current answer is correct, and $score$ is the score the learner would get if it submitted the exercise in its current state.

This should, in effect, simulate how an individual learner would act while answering an exercise providing us with a useful benchmark of how well the feedback system learns a learner's feedback preferences. Now that we have explained the process by which the simulated learners answer an exercise we will now explain the parameters chosen for the simulated course, why they were selected and the range of values tried.

### 8.1.2 Simulated Courses

The context being simulated was the use of Shufti as an instructional aid to students learning mammography. The classes of 10 simulated learners completed exercises and learned in a simulated manner as described in the next section. The number of exercises the learners completed was 450, or 5 a day for 90 days. This represents a reasonable course load and number of exercises completed.

### 8.1.3 Simulated Learners

The simulated learners are essentially a set of values and behaviours which approximate the actions of a real mammography student. They slowly improve over time much as a real student would, and are able to make mistakes and become impatient, just as real students do. Additionally, each student is unique in how it will react to various stimuli and situations.

In order for learners to be realistic simulations they have to produce practical answers to the exercises similar to those produced by real students and they have to improve their performance over time in a reasonable manner similar to that of real learners slowly improving over a course.

The learners must also have their own unique preferences for feedback, replicating how some students thrive on praise and some simply want correction or no interaction at all.

And finally, the learners have to be able to decide when they are done an exercise and wish to submit it. Over this next section we enumerate each of these features and the parameters involved, their purpose, how we chose the values for them and why we chose them.

The first and perhaps most important feature of the simulated learners is their ability to answer questions in a realistic manner as described in Section 8.1.1. The key learner parameter for this feature is $p_{wrong}$ which represents a learner's tendency to make mistakes. When a learner is generated, $p_{wrong}$ is initialized randomly to a normally distributed value with a mean of 0.3 and a standard deviation of 0.1, having a maximum of 1 and a minimum 0. These numbers were chosen to replicate the average of 60% correct that a reasonable student in a course should produce. The standard deviation is chosen to account for the proportion of students which should be above average and below average.

Learners also have to improve, and to simulate this, each time the learner submits an exercise its $p_{wrong}$ is permanently changed to the result of $p_{wrong} \cdot learningRate$. This progression is chosen to resemble the diminishing returns which learners experience as they progress though a subject. Initially their

progress is quick as they fill large gaps in their knowledge, but later more subtle mistakes are made and need to be corrected resulting in a reduction of gains.

In our case we chose to generate the learner's learning rates from a Normal Distribution with a mean of 0.998 and a variance of 0.001, limiting the final result to a maximum of 0.999 and a minimum of 0.994 which served to remove unrealistic outliers. The values for mean and variance were chosen to have the average learner be approximately 80% correct by the end of the simulated course ends much as a typical real student would be. For an idea of the learner's progression see Figure 8.1.

The most critical feature to our evaluation is for each learner to have their own unique reaction to various feedbacks. This is achieved by temporarily perturbing $p_{wrong}$ by multiplying it by the $feedbackEffect_n$ when feedback $n$ is issued to the learner during a simulated exercise. These feedback effects are determined once, randomly at the time of creation of the learner. They are distributed on a Normal Distribution with a mean of 1.0 and a standard deviation of 0.4. The mean in this case is chosen to give equal likelihood to a student liking or disliking a given feedback. The variance is chosen to represent the possible strength that a feedback can have on a student's experience. This means that the naive approach of issuing random feedbacks will have the same approximate result as issuing no feedback at all, the system can only improve the learner's performance if it correctly determines the individual learners preferred feedback.

Finally, learners have their submission values $timeWeight$ and $scoreWeight$ which together control when they submit an exercise. These two parameters are distributed Normally with a mean of 7 and standard deviation of 2 and a mean of 50 and standard deviation of 5 respectively. This results in the simulated learners taking an average of 10.3 cycles of the answering process to decide to submit an exercise.

The combined features described in this section should result in a learner which is both realistic and enables us to evaluate our feedback selection system.

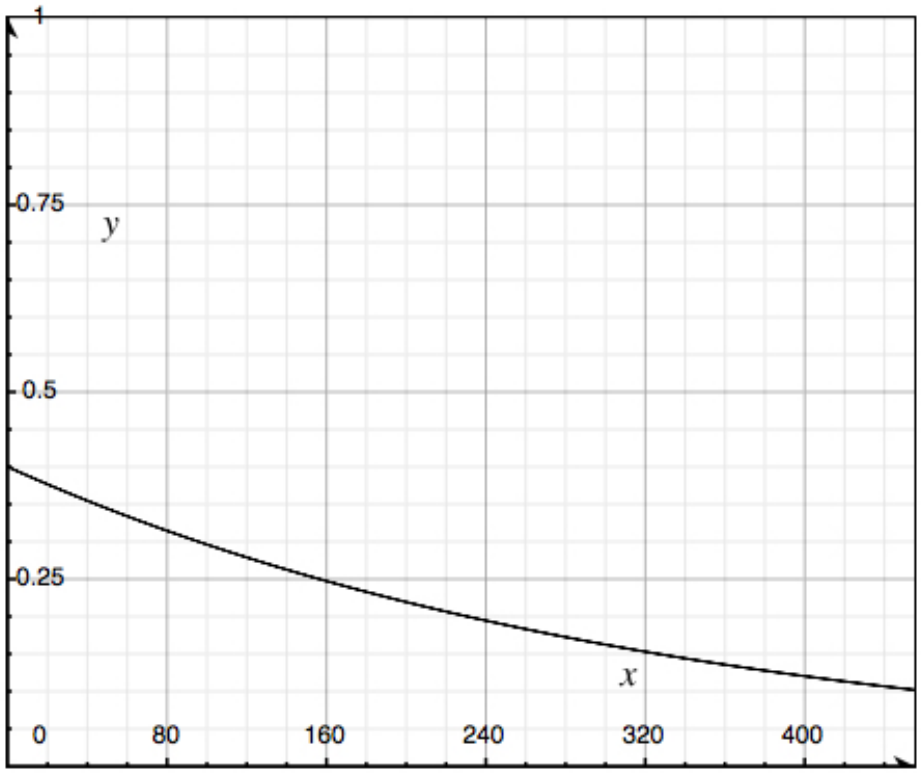


Figure 8.1: A typical simulated learner's $p_{wrong}$ progression over exercises

## 8.2 Algorithm Selection

Once we had a working simulator we were able to use it to select the best reinforcement learning algorithm to underpin our feedback system. This is important as some RL algorithms are more well suited to some problems than others. We will now compare the three algorithms discussed in Section 7.1.1

Now that we had defined what we are comparing we had to decide how we were going to compare them. A scenario was chosen in which the algorithms would be evaluated over four runs totalling 450 exercise each with a total of ten students. The feedbacks available would be two positive and two negative with an additional no feedback response possible. To determine the which algorithm performs the best we would average the class averages of four runs for each method. As a test of significance we used a two tailed unpaired t test to compute a p-value of each result vs. the baseline of no feedback selection algorithm.

| Algorithm | Score Out Of 450 | Proportion Correct | p-value |
|---|---|---|---|
| None | 271.13±15.58 | 0.603±0.0346 | – |
| SARSA | 336.29±6.45 | 0.747±0.0143 | 0.0005 |
| Q-Learning | 309.20±17.05 | 0.687±0.038 | 0.0290 |
| SARSA-$\lambda$ | 348.34±2.58 | 0.774±0.005 | 0.0001 |

Table 8.1: The performances of the various algorithms

After running these tests, the results of which can be found in Table 8.1, we found that SARSA-Lambda proved to be the highest performing method, beating the closest competitor by 3 percentage points and the base line by 17. The results also appear to be very significant with all methods clearing the common p-value significance threshold of 0.05. This promising result led to SARSA-Lambda being chosen for further tuning. As for why SARSA-Lambda performed the best we can hypothesize that on-policy methods such as SARSA and SARSA-Lambda will more quickly account for their exploration taking them into incorrect actions. This is due to their use of the policy specified Q-value in their update rules (unlike off-policy methods, such as Q-learning). This speed of learning incorrect actions is important because if the simulator

takes the wrong action it can drastically worsen the learner's performance and adversely impact the agent's reward.

SARSA-Lambda out performed normal SARSA because its eligibility traces sped up learning by ensuring past choices received credit more quickly than with non-tracing methods.

## 8.3 Parameter Turning

Once we selected our algorithm (SARSA-Lambda) we explored the possible values for its parameters. The possible tuning values are the learning rate $\alpha$, the reward discount $\gamma$, the exploration rate $\epsilon$, and the tracing value $\lambda$. Together these values can influence results in a positive or negative manner and we wanted to attempt to yield a performance gain. Using the same method as in the algorithm selection Section (8.2) we took an average of four runs to ensure that the randomness in the simulation does not have a disproportionate effect on the results. We also retain the two positive, two negative and one non feedback options for the feedback system to select from.

| $\epsilon$ | $\alpha$ | $\gamma$ | $\lambda$ | Average Score | p-value vs. baseline |
|---|---|---|---|---|---|
| 0.05 | 0.3 | 0.2 | 0.8 | $0.712 \pm 0.047$ | 0.0169 |
| 0.05 | 0.3 | 0.2 | 0.3 | $0.703 \pm 0.037$ | 0.0141 |
| 0.05 | 0.5 | 0.2 | 0.3 | $0.669 \pm 0.027$ | 0.0384 |
| 0.05 | 0.15 | 0.2 | 0.3 | $0.776 \pm 0.013$ | 0.0002 |
| 0.05 | 0.15 | 0.2 | 0.5 | $0.699 \pm 0.025$ | 0.0083 |

Table 8.2: The performance of SARSA-Lambda with various parameters

As illustrated in Table 8.2 the highest results yielded are $0.776 \pm 0.013$. These were produced using SARSA-lambda with an $\epsilon$ of 0.05, an $\alpha$ of 0.15, a $\gamma$ of 0.3, and a $\lambda$ of 0.3.

## 8.4 Results Discussion

Following algorithm and parameter selection, performance their performance implications must be examined in more detail. To do this we compared the existing baseline with runs using our best possible parameter values. We also

evaluated the performance implications of using a larger set of feedbacks for the algorithm to issue.

As shown in the previous section, using parameter values of $\epsilon$ at 0.05, $\alpha$ at 0.15, $\gamma$ at 0.2, and $\lambda$ at 0.3 we achieved an average learner score of nearly 78% with a variance of 0.013 almost 18 percentage points better than our naive baseline where the students performed at an average of 60% with a standard deviation of 0.034. A two-tailed unpaired t-test of this result yielded a p-value of 0.0002, much lower than the common significance threshold of 0.05 making this a very significant result.

To see the effects of using a larger set of feedbacks we ran a set of tests using different amounts of available feedbacks ranging from two to six feedbacks of each polarity. We averaged sets of 4 runs with each learner completing 450 exercises, much as in Section 8.2. The results of this can be seen in Table 8.3.

| Positive Feedback | Negative Feedback | Average Score |
|---|---|---|
| 0 | 0 | 0.603±0.035 |
| 2 | 2 | 0.776 ±0.013 |
| 3 | 3 | 0.766 ±0.022 |
| 4 | 4 | 0.749 ±0.030 |
| 5 | 5 | 0.788 ±0.013 |
| 6 | 6 | 0.780 ± 0.011 |

Table 8.3: The performance of SARSA-Lambda with various parameters

As can be seen in Table 8.3, using different numbers of positive and negative feedbacks does not have a large impact on the average scores. In this instance, having five positive and negative feedbacks available achieved the best result 0.788 but that is only marginally better than having a set of two feedbacks available (0.776). And using three feedbacks is marginally worse than using two (0.766). This should mean that the feedback system can perform in a variety situations with varying amounts of feedback possibilities. Incidentally, it was not necessary to test using differing numbers of positive versus negative feedbacks as only appropriate feedbacks are selected meaning the choice of a particular positive feedback is largely divorced from the choice of a particular negative feedback and vice versa..

Ultimately using SARSA-Lambda with the appropriate parameters achieves a significant gain over the baseline, suggesting that the use of the Shufti ITS in a real-world learning situation would be of significant value to students in the medical imaging field.

# Chapter 9

# Future Applications

The innovations within Shufti are not limited to just the diagnosis of mammograms. Other medical imaging tasks are similar enough to mammography that the techniques in Shufti are still useful. Other medical imaging fields where these techniques are applicable include dermatology and pathology. In pathology, the goal is to analyze biopsies for diagnosis. This is done by finding evidence of a condition in the biopsy. This process is similar to locating lesions in mammograms. Therefore, it is possible to use the same structure as Shufti's grid or polygon questions in a pathology ITS. This would also allow us to use Shufti's feedback system with minimal modification. Moreover, as Shufti's gamification techniques are already domain-independent, they are easy to utilize in other medical imaging ITSs.

To illustrate the adaptability of the techniques in more depth, we will now examine them in the context of dermatology. Dermatology is the diagnosis and treatment of skin conditions and diseases [14]. For our proposes, we will focus on the diagnosis section. To diagnose a skin disease, melanoma in particular, the first step is a high-level visual examination. To do this diagnosis, a dermatologists examines the skin area looking for evidence as to what the condition is.

If we pose this problem as locating evidence of the condition in an image, we can simply see the similarity of the problem to Shufti's grid and polygon exercises. Both involve locating regions of interest within an image. This exercise also has the same domain attributes as mammography in Shufti. The

domain is verifiable using an expert-produced gold standard. Neither does it have formal theorems as it relies on the a medical professional's judgment. The domain is also made up of discreet sub-problems in that there are no overlapping steps in performing a diagnosis. These similarities should enable the use of the same gamification and feedback selection techniques as in Shufti.

# Chapter 10

# Conclusion

Analysis of medical images is an important medical task requiring significant training. At the moment students take part in a practicum in which they follow the cases of an established medical professional. This practicum model has many desirable attributes such as providing one-on-one tutoring to the students. Unfortunately the cases seen by the students during their practicum are limited to those seen by their instructor. An Intelligent Tutoring System can provide access to many more cases improving the variety seen and allowing for practice on rare cases. To this end we have created Shufti a mammography Intelligent Tutoring System.

To create Shufti we have examined existing Intelligent Tutoring Systems in well-defined and ill-defined domains. We examined the concept of feedback, and produced a framework to categorize it. We added modern gamification techniques to shufti, to increase student engagement.

It is still desirable to retain the benefits of the existing practicum system, in particular the advice and guidance a tutor can provide. To do this we have created a new reinforcement learning based feedback selection technique. This method characterizes the system issuing feedback as a reinforcement learner selecting an action, and uses the state of the exercise as the reinforcement learning agents state signal.

Our study culminated in the investigation of the effectiveness of the feedback selection system in a course simulator, an investigation which yielded promising results. Which we hope to later use in other medical imaging tasks,

perhaps even extending the technique devised into other fields of instruction which are verifiable. In closing, the data suggests that the Shufti ITS could prove a useful addition to the field of mammography education, Despite these encouraging results, each of the topics covered warrants further exploration with particular focus on the simulation of learners and the feedback selection system.

# Bibliography


[1] David Abraham, Liz Crawford, Leanna Lesta, Agathe Merceron, and Kalina Yacef. IMEJ Article - The Logic Tutor: A Multimedia Presentation, 2001.

[2] Vincent Aleven. Rule-Based Cognitive Modeling for Intelligent Tutoring Systems. In *Advances in Intelligent Tutoring Systems*, volume 308, chapter 3, pages 33–62. 2010.

[3] Vincent Aleven, Bruce M. McLaren, and Jonathan Sewall. Scaling up programming by demonstration for intelligent tutoring systems development: An open-access web site for middle school mathematics learning. *IEEE Transactions on Learning Technologies*, 2(2):64–78, 2009.

[4] Benjamin S. Bloom. The 2 Sigma Problem: The Search for Methods of Group Instruction as Effective as One-to-One Tutoring, 1984.

[5] Jacqueline Bourdeau and Monique Grandbastien. *Modeling Tutoring Knowledge*, volume 308, chapter 7, pages 125–143. Jacqueline Bourdeau and Monique Grandbastien, 2010.

[6] Elizabeth O. Bratt. Intelligent tutoring for ill-defined domains in military simulation-based training. *International Journal of Artificial Intelligence in Education*, 19(4):337–356, 2009.

[7] I Cheng, F Chen, S Rodrigues, O.G. Pañella, L Vicent, and A Basu. Intelligent Games for Education - An Intention Monitoring Approach based on Dynamic Bayesian Network. 2010.

[8] Irene Cheng, Nathaniel Rossol, and Randy Goebel. Self-Tutoring , Teaching and Testing : An Intelligent Process Analyzer Computer and Information Science , University of Pennsylvania Dept . of Computing Science , University of Alberta. In *2008 Eighth IEEE International Conference on Advanced Learning Technologies*, volume 1, pages 746–750. IEEE, 2008.

[9] Rebecca Crowley, Olga Medvedeva, and Drazen Jukic. SlideTutor: A model-tracing Intelligent Tutoring System for teaching microscopic diagnosis. *Artificial Intelligence in Education: Shaping the Future of Learning Through Intelligent Technologies*, 97, 2003.

[10] Philippe Fournier-Viger, Roger Nkambou, and Engelbert Mephu Nguifo. Building Intelligent Tutoring Systems for Ill-Defined Domains. In *Advances in Intelligent Tutoring Systems*, volume 308, chapter 5, pages 81–101. 2010.

[11] James P. Gee. Deep Learning Properties of Good Digital Games How Far Can They Go? *Theories and Mechanisms: Serious Games for Learning*, pages 63–80, 2009.

[12] Ilya M. Goldin, Kevin D. Ashley, and Rosa L. Pinkus. Introducing PETE : Computer Support for Teaching Ethics. *Proceedings of the 8th International Conference on Artificial Intelligence and Law*, pages 94–98, 2000.

[13] M. Heath, K. Bowyer, D. Kopans, R. Moore, and P. Kegelmeyer. The digital database for screening mammography. *Proceedings of the Fifth International Workshop on Digital Mammography*, pages 212–218, 2001.

[14] Brian Hibler, Qiaochu Qi, and Anthony Rossi. Current state of imaging in dermatology. *Seminars in Cutaneous Medicine and Surgery*, 35(1):2–8, 2016.

[15] George Hripcsak and Adam S. Rothschild. Agreement, the F-measure, and reliability in information retrieval. *Journal of the American Medical Informatics Association*, 12(3):296–298, 2005.

[16] Stuart Johnson and Osmar R Zaïane. Deciding on Feedback Polarity and Timing. In *Education Data Mining*, 2012.

[17] Stuart H Johnson and Osmar R Zaiane. Intelligent feedback polarity and timing selection in the Shufti Intelligent Tutoring System. In *International Conference on Computers in Education*, 2012.

[18] C. Lynch, K. Ashley, and V. Aleven. Defining ill-defined domains; a literature survey. In *Proceedings of the Workshop on Intelligent Tutoring Systems for Ill-Defined Domains at the 8th International Conference on Intelligent Tutoring Systems.*, 2006.

[19] Erica Melis. ActiveMath : An Intelligent Tutoring System for Mathematics. *In Seventh International Conference Artificial Intelligence and Soft Computing' (ICAISC), volume 3070 of LNAI*, pages 91—-101, 2004.

[20] Antonija Mitrovic. Modeling Domains and Students with Constraint-Based Modeling. In *Advances in Intelligent Tutoring Systems*, volume 308, chapter 4, pages 63–80. 2010.

[21] Alvin I. Mushlin, Ruth W. Kouides, and David E. Shapiro. Estimating the accuracy of screening mammography: A meta-analysis. *American Journal of Preventive Medicine*, 14(2):143–153, 1998.

[22] Roger Nkambou. Modeling the Domain: An Introduction to the Expert Module. In *Advances in Intelligent Tutoring Systems*, volume 308, chapter 2, pages 15–32. 2010.

[23] Roger Nkambou, Jacqueline Bourdeau, and Riichiro Mizoguchi. *Advances in Intelligent Tutoring Systems*, volume 308. 2010.

[24] Harold F. O'Neil, Eva L. Baker, and Ray S. Perez. *Using Games and Simulations for Teaching and Assessment*. Routledge, 2016.

[25] Daniel Perelman and Dan Grossman. Test-Driven Synthesis for Automated Feedback for Introductory Computer Science Assignments Categories and Subject Descriptors. 2014.

[26] Harry Pople. Heuristic Methods for Imposing Structure on Ill-Structured Problems: The Structuring of Medical Diagnostics, 1982.

[27] John Self. Theoretical Foundations for Intelligent Tutoring Systems. *Journal of Artificial Intelligence in Education*, 1(45):3–14, 1990.

[28] Tarja Susi, Mikael Johannesson, and Per Backlund. Serious Games An Overview. *Elearning*, 73(10):28, 2007.

[29] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[30] Mary Ulicsak. Games in Education: Serious Games. *A FutureLab Literature Review*, page 139, 2010.

[31] Bram van Ginneken, Mikkel B. Stegmann, and Marco Loog. Segmentation of anatomical structures in chest radiographs using supervised methods: A comparative study on a public database. *Medical Image Analysis*, 10(1):19–40, 2006.

[32] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[33] Gerhard Weber and Antje Mollenberg. ELM-PE: A Knowledge-based Programming Environment for Learning LISP. *Ed-Media '94*, pages 557–562, 1994.

[34] Beverly Park Woolf. *Student Modeling*, volume 308, chapter 8, pages 267–279. 2010.

[35] Kevin Yee. *Pedagogical Gamification: Principles of Video Games That Can Enhance Teaching*, volume 32. 2013.