University of Alberta

IGOT : an Intelligent GO Tutorial

by

Jean Leroux ©

A thesis
submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree
of Master of Science

Department of Computing Science

Edmonton, Alberta
Spring 1990

ISBN 0-315-60189-2

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Jean Leroux
TITLE OF THESIS: IGoT : an Intelligent *Go* Tutorial

DEGREE: Master of Science
YEAR THIS DEGREE GRANTED: 1990

(Signed) [illegible]

Permanent Address:
Mauvoy
41370 Marchenoir
FRANCE

Date: May 1, 1990

A game of Go is symbolic of the gradual occupation of our planet by the human race. Its sides are the coasts, washed by oceans and seas. The corners can be compared to islands or peninsulas. Those parts having a greater coastline are more easily defended.

The central part of the map corresponds to the centers of continents where the inhabitants have no outlet to the sea. In the beginning, men were very few and the families or tribes had all the territory they desired without need for offensive or defensive action. They lived in a state of nature. But with the multiplication of human beings began the first struggles for the appropriation of the best places along the rivers and sea coasts. As the game develops and bases have been consolidated, advance to the interior is begun. The occupation of this territory is rendered permanent by the formation of live masses.

When the war ends peace treaties are made. All territories on the map are occupied. In one place we find large or small masses that have definitely won their territory, in another, masses living side by side respecting the rights of their neighbors whom they can never hope to dislodge.

We have not arrived, in our world, at the state of finality achieved at the end of a game of Go.

From *The game of Wei-Chi* by Count Daniele Pecorini and Tong Shu. (published in 1929)

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled IGoT : **an Intelligent Go Tutorial** submitted by **Jean Leroux** in partial fulfillment of the requirements for the degree of Master of Science.

[illegible]
(Supervisor)

[illegible]

[illegible]

[illegible]

[illegible]

Date: April 30, 1990

To My parents,
My friends,
The *Go* players,
and Alison's toe.

Abstract

This thesis explores the domain of Intelligent Tutorial Systems, closely linked to the field of artificial intelligence. We focus on one of the hottest issues in the ITS literature: ***Student Modeling.*** It consists of keeping an explicit representation of the student in the tutoring system and exploiting it to make lessons adapted to the student and therefore conveying intelligent features to the system.

The originality of this document is to propose the design of the first ITS for the old Chinese game of *GO*. The *GO* knowledge is divided into ***units of knowledge*** that constitute a measure of the student's performance and they together form the student model.

Every lesson is made out of *GO* problems and we propose a method to evaluate the answer of the student based on the *GO* rating system by transforming a problem into a game. This operation allows the system to evaluate the rating of the student at any time. Major deficiencies in the *GO* knowledge can be detected to help the creation of new lessons adapted to the student.

The system is not rigidly attached to the student's representation and enables different degrees of freedom in the creation of a lesson. The automatic option sets up parameters according to the student model. The manual option allows all the parameters to be modified to create a fully customized lesson independent of the student model.

The system is developed on an *IBM-XT* with the *Clipper* language and the database is built with the *DBase III plus* protocol.

After a brief introduction to ITS and Student Modeling we present the design and the implementation of IGOT (Intelligent *GO* Tutorial). Then the last chapters are devoted to the tests and results, and some suggestions for the next version of IGOT.

Acknowledgements

I would like to thank my supervisor, Professor Tony Marsland, for his support throughout this research. I would like to thank the *Go* expert, Dr. Chuck Elliott for his guidance and helpful criticism during the project. I extend this thanks to his colleagues of Kakari Systems where I always had access to a computer and a comfortable technical support from the programmers, especially Brenda Hawkins and Jeff McKay. I would like to thank Alison McCreary who helped me to correct the English and improve the style of the first draft. Many thanks to the *Go* players who collaborated for the test and can now understand its meaning with this document they are about to enjoy.

Contents

List of Figures

List of Tables

Chapter 1

Student Modeling in ITS

1.1 Intelligent Tutorial Systems

1.1.1 History

The increasing technological sophistication of our society requires people to master an increased variety of complicated subject material, to master an increased set of sophisticated skills, and to perform these skills at higher standards of performance ([O'Neil 81], page 2). In the early 60's this need gave birth to a new field: computer-based instructional systems. This section presents the evolution of these systems during their short history and the hottest issues.

The first generation of computer-based instructional systems appeared in education and was known as *Computer Aided Instruction* (CAI). Then in the mid 60's, Uhr and his collaborators implemented *adaptative systems* that selected problems of arithmetic according to the student's overall performance [Uhr 69]. The Models of the student were based more on *parametric summaries* of behavior rather than *explicit* representations of knowledge. **Adaptative CAI** evolved to **Intelligent CAI** which extended their power and accuracy.

The last generation of computer-based instructional systems expands their domain of applicability and leads to a more general acronym, **ITS**, standing for *Intelligent Tutorial Systems.* They use many concepts of **Artificial Intelligence** that distinguish them from more traditional approaches. Etienne Wenger, in his excellent survey of Tutoring Systems, explains clearly the role of Artificial Intelligence in Instructional Systems [Wenger 87].

1.1.2 The Encoding of Knowledge in ITS

The traditional instructional systems embody both *the domain* and the *pedagogical knowledge of experts.* The expertise to be conveyed is contained in prestored presentation blocks, sometimes called *frames*, which are designed by the expert teacher and displayed to the student under given conditions.

In ITS the purpose is to encode the knowledge itself rather than the decisions that result from some pedagogical expert knowledge. The instructional interactions are performed by programs that make decisions by reference to the knowledge. The encoding of the expertise rather than the decisions of experts allows the system to make decisions not anticipated by the experts.

1.1.3 Knowledge Communication

Knowledge communication is the *"ability to cause and support the acquisition of one's knowledge by someone else via a restricted set of communication operations"* ([Wenger 87], page 7). The set of communication operations can include a language, graphic screens, a set of exercises, but not a direct reference to the encoded knowledge.

The notion of knowledge communication implies a view of *pedagogic activities* as a capability of intelligent tutorial systems. It involves most of the research issues in ITS and the literature generally distinguishes four components in the task of knowledge communication ([Wenger 87], pp 14–22), ([Spada 89], page 487):

The Object of Communication: The system contains an EXPERT MODULE that is the source of the knowledge to be communicated. It generates tasks and questions as well as explanations and responses to the students and serves as a standard for evaluating the students' performance. This module can establish a measure to grade and compare knowledge and thereby measure the students' knowledge.

The Recipient of Communication: Good human teachers have an ability to perceive the students' view and to adapt their behavior accordingly. The process of communication requires a certain understanding of the *recipient.* In ITS the students are represented with a STUDENT MODEL. This Model is meant to reflect most of the aspects of the students' knowledge that can have repercussions for their performance and learning. This is the object of Section 1.2.

The Skill of Communication: The PEDAGOGICAL KNOWLEDGE is often deeply buried in the various pieces of the code that control the ITS. A research issue consists of the explicit representation of the pedagogical knowledge where decisions can be derived from a set of rules representing the system's pedagogical expertise.

The Form of Communication: The INTERFACE is the module that takes care of the final form of communication. It translates in both directions between the system's internal representation and the *interface language*, understandable by the students.

1.2 The Student Model

1.2.1 The Ideal Student Model

In the arena of ITS, a review of the literature shows an increasing interest for *student modeling* [Spada 89], ([Greve 89], page 151), ([Nakhoon 89], page 258), ([Witsc 89], page 599), [Brecht 88]. The ideal student model should include all the aspects of the students' behavior and knowledge that can influence their learning. Such a task is particularly difficult because the number of aspects to consider is too big. Moreover the *communication channel* is restricted to the screen and the keyboard. It can't interact with other sources like voice effects or facial expressions. Therefore systems designers only use **partial models** that provide some partial representations of the students with essential information for the specific tutorial.

1.2.2 The Tutorial Dialogue

One can imagine the tutorial dialogue as an interaction between two learners ([Spada 89], page 487). **The student** acquires knowledge about the domain, instructed or coached by the tutor. **The Tutor** acquires knowledge about the student using as data the student's reaction.

Good tutors should be like good teachers. They should adapt their *instructional measures* to the knowledge state of the learners. This purpose implies three goals for student modeling:

- **To Build** the knowledge state of the student.
- **To Compare** his knowledge state to a criterion model and therefore evaluate the student.
- **To Decide** what the tutor has to teach next.

These goals allow *self-adapting systems* a constructive process of knowledge acquisition by updating the student models during the learning process and adjust their instructional measures accordingly.

1.2.3 Student Modeling

Students have to acquire knowledge in assimilable increments, and the scope of their expertise will grow progressively. Such differences in scope have prompted the representation of a model's scope as an *overlay*: expertise is decomposed into independent components that become orthogonal dimensions of variability in knowledge and a system of markings is superimposed on the model of expertise to indicate the level of mastery of these individual units of knowledge. The subsets thus defined inherit the substantial characteristics of the model. Overlays provide a simple mechanism to determine candidate areas for pedagogical action ([Wenger 87], pp 345–355).

Under the overlay paradigm, variations in knowledge states result from incompleteness, but never from incorrectness. However, there is ample evidence that inappropriate behavior often stems from existing but incorrect versions of pieces of knowledge or even from incorrect beliefs that bear no relation to the correct expertise, so that information about which deviations to expect in a domain plays an important role in a tutor's ability to communicate knowledge.

The study of epistemic deviations observed in a specific domain has usually been called a *theory of bugs*. Because it is important to perceive the student's knowledge exactly, such theories have received a fair amount of attention. Three approaches are famous: enumerative theory, reconstructive theory and generative theory. The first theory consists in building a library of commonly observable deviations, which are individually recognized as underlying the student's behavior. The reconstructive theory attempts to reconstruct internalized bugs on the basis of observed errors. The last theory not only detects the errors like the previous ones, but also tries to give an explanation of the bugs in term of their generation process.

Besides the theories of bugs, building a student model raises different problems of design:

Granularity: How *detailed* should a partial student model be? What *relevant* information should be represented?

Dynamic: What structure to choose to reflect the changes of the students' knowledge in the specific domain? How to do these changes automatically ([Spada 89], page 488) ?

Learning Theory: What domain specific theory of learning has to be added to the domain representation so that it allows the domain knowledge to be accessed and manipulated according to the dynamic of the student model ([Spada 89], page 488) ?

Degree of Freedom: What liberty the system should give to the student? To what extent can a tutor place constraints on the freedom of the learner? Megarry raises this question when he examines the contrast between intelligent tutoring methods and the potential for hypermedia[1] [Megarry 89].

> *The role of the computer should be organizing and representing knowledge to give the user easy access and control, rather than trying to create a model of the learner and seeking to prescribe her route through it (...) a false trail has been laid by intelligent tutoring systems that try to create a model of the student (...) to treat the learner as a dumb patient and the computer system as an omniscient doctor is both perverse and arrogant...*

Whether the idea of an explicit student model has to be rejected, we must wonder how to give the user *easy access and control* to the knowledge domain.

[1]Hypermedia is a general term. Through a direct access with information items they permit the rapid and efficient access to further relevant information. They are designed for information presentation, exploratory browsing, information creation, organization, management. While the material is limited to static text for **Hypertext**, video, computer graphics, animation, sound, and any technologically feasible form of presentation are all included in **Hypermedia** [Hammon 89], [Conklin 87].

1.2.4 Summary

Building an ITS is an interdisciplinary enterprise that can involve researchers from a variety of backgrounds such as Cognitive Psychology and Artificial Intelligence, but with a common interest. The major issues consist of defining a model of the knowledge and a model of the student. Good teaching strategies must be found to make the tutorial dialogue efficient, as outlined in Section 1.1.3.

One main focus of Cognitive Science lies on this issue of how to construct a system which has the special feature to adjust its behavior to the user in a sophisticated way. One answer is student modeling. Findings of Cognitive Psychology and tools of Artificial Intelligence are combined to assess the students' knowledge and the learning processes they are subjected to, while working with the system. Several approaches towards student modeling are possible, namely overlay models, enumerative diagnosis systems, and generative models based on theories of knowledge acquisition. The errors of the students are used to understand their learning process. The systems don't consider the students as empty experts, but as novices with beliefs and deviations from the correct knowledge ([Wenger 87], pp 345–355).

Chapter 2

The Design of $\mathrm{I^{G}_{O}T}$

2.1 Motivation

ITS are no longer restricted to the domain of education. Their principal fields of applicability are medicine (MYCIN[1]), engineering (STEAMER[2]), mathematics (EXCHECK[3]), natural language (SOPHIE[4]). There is little literature concerning ITS for classical games like chess, G_O, backgammon and othello. A famous board game in the domain of ITS is WEST ([Sleeman 82], page 85). It is designed to give students practise in arithmetic, but its pedagogical objectives don't come from the game. Therefore the approach of the designer is to create an ARTIFICIAL GAME as an interface for knowledge communication. This game is typically used by one student

[1]MYCIN is an expert consultation system in the diagnosis of bacterial infections.

[2]STEAMER is a manipulable simulation of a steam plant for training engineers that will operate large ships.

[3]EXCHECK, 1981, is a learning environment to understand proof procedures.

[4]SOPHIE (SOPHisticated Instructional Environment) provides a reactive learning environment in the domain of electronic circuits. The students try their ideas, have them criticised, and receive advice.

playing against the computer's expert[5].

When one builds an instructional system, one needs the support of a human expert in the domain although certain tutorials have been built by researchers without a real expert. We had the opportunity to discuss the possibility of a tutorial for the game of *Go* (see Appendix A) with one of the best *Go* teachers in Canada: Dr. Chuck Elliott. His twenty years of teaching have helped him to understand some mechanisms of the process of learning through the various experiences he had with players. More than the knowledge of a *Go* player, Dr. Elliott has developed some pedagogical skills in this area. In our discussions it appeared that the *Go* world doesn't have enough teachers for the number of players. There is no instructional system so far to help *Go* clubs in need of teachers and experts. There isn't even a database of *Go* problems where the knowledge is organized and evaluated by an expert. The need for this type of *Go* library is obvious in the *Go* world.

Typically *Go* teachers work with problems that they try to analyze and solve with students. Building an ITS that would propose lessons of problems seems the most natural way to communicate skills. But an ITS needs some measure to evaluate the performance of the students. No adaptative instructional system can be built otherwise. There is a notion of measure in the game of *Go*. It is a handicap system (see Appendix B) that enables one to evaluate the strength of each player. The problem is that players are rated according to the results of their games with other players. Our discussion with Dr. Elliott has lead us to propose a method to transform a problem into a game played against the computer and to use the handicap system

[5]It is also possible for two students to play against each other, in which case differential models are constructed for each student, enabling coaching for both players.

as a measure of the efforts of the student. This method is based on the relationship between the level of expertise of a player and the time needed to solve a problem. It is fairly well accepted that this relation is linear or at least linear for a wide range of strengths. Once we can appreciate the answer to a problem and grade it with a solid system, we can build an ITS.

We use the overlay paradigm (see Section 1.2.3) to indicate the variations in knowledge states resulting from incompleteness. The expert can break the Go knowledge into units of knowledge for which the handicap system should reflect the level of mastery. No theory of bugs has been discussed for this first version of the tutorial.

This thesis is the first step towards building an ITS for existing board games and the first case study in Go. Although an ITS for chess is under investigation by Gadwal [Gadwal 89], nothing has been done for the old Chinese game of Go. The abundant literature provided by the ISHI Press [Ishi] constitutes a good source of commented games and problems to build the Expert module of an ITS. Adapting a tutorial system to the state of knowledge of a student is fascinating for Go players that desire to increase their level. When they want to work on a specific area of knowledge, players have to open several books and the difficulty of the problems encountered may not be appropriate to their level, because each book is organized differently for a particular pedagogical purpose. We wish to provide a tool that does the same thing but with an easier *access and control* to the knowledge. We call it **IGoT (Intelligent Go Tutorial).**

2.2 The Pedagogical Objectives

The definition of the pedagogical objective is an important preliminary point. It must take into account the learner's level, or at least that of a group of learners. The main objective has to be decomposed into sub-objectives to elaborate the ultimate goal ([Abayomi 89], page 56). The purpose of IGoT is to provide Go players with additional knowledge that will help them play better games. We must present the game a little deeper to define sub-objectives. Understanding the rules of the game (Appendix A) is easy. On the contrary, mastering strategies and concepts that characterize a good player requires more work. Amateurs typically go through different books of problems when in need of improving their skills.

The game of Go is a model of war and the better player is often the one that can coordinate all his campaigns [Kierulf 89]. Many **concepts** or tactical moves are involved in these campaigns. They are classified under **categories**, like *Life and Death, Endgame, Fuseki...* (see Appendix C for a complete list) according to the type of strategy involved and the progress of the game[6]. The Go literature distinguishes about six categories while the list of concepts is a lot longer.

Our objectives for IGoT are:

- **To maintain a balance of the students' level of knowledge between the different categories.** A good player should have a well balanced knowledge of the concepts, all of which appear during the progress of a game. Typically, players lose a game just because they don't play the *Endgame* well even after

[6]The first stones involve some tactical moves of *Opening* while the last stones are probably typical moves of *Endgame*.

an excellent beginning.

- **To raise the overall level of *Go* knowledge during the learning process to a good amateur level.**
- **To provide a measure of the students' strength that can reflect their behavior in a real game.** *Go* students need a measure to feel that the time they spend with the tutorial system for training will pay off. They generally have more problems noticing an improvement in their skills than do sport students, who can feel directly the benefit of training. Measuring the students' knowledge is challenging.
- **To organize *Go* lessons according to various criteria (category, concepts, range of difficulty...).** The system should be able to adapt the lessons to the students. A certain degree of liberty must characterize I*Go*T though, so that the students don't feel any constraint imposed by a rigid system. We must have the possibility of organizing lessons independently of a student model, for activities such as *Go* lectures or publishing.

2.3 The Pedagogical Strategy

2.3.1 Theory of Knowledge Acquisition

The somewhat classical strategies for the acquisition of knowledge range from the most simple, like *memorization*, to the most complex, like *learning by reasoning*, or most interesting: *learning by discovery* [Abayomi 89]. Various strategies are in use to learn by discovery. One of them is simulation. STEAMER for instance, is a manipulable simulation of a steam plant for training engineers that will operate on large ships ([Wenger 87], pp 80–86). Recognition is another mode of discovery. Students try their ideas, have them criticized and receive advice as in SOPHIE ([Wenger 87], pp 59–70). We attempt to follow this last strategy in IGoT.

One could imagine the ideal *Go* tutorial as an excellent *Go* program providing comments of variable depth on each stone played during a game. This would underline the weaknesses of the player, suggest the next move, and have some tactical moves of previous games replayed to make sure the student doesn't repeat the mistake. The first necessary condition is to build a *Go* program.

The world of Computer *Go* [7] is young and there are many open questions. Some of the best *Go* programs [Kierulf 89], [Shira 89], play at a level of about 13 kyu[8]. In his talk in Edmonton during the WCCC[9] in 1989, J. McCarthy[10] said that in terms of strength, if computer chess programs are three levels behind the best human players, then the gap corresponds to twenty of these levels for computer *Go* programs. This statement does not encourage us to continue in the direction of an ideal *Go* tutor.

A different approach is based on the typical training *Go* players go through. It consists in displaying typical problems, analyzing the answers, then giving a brief explanation. We believe that going through a series of similar problems is necessary to learn a new concept and apply efficiently this new skill in a real game. A Chinese

[7]Computer *Go* is the area of research devoted to the development of techniques for building *Go* programs.

[8]See **Appendix B** to understand the notion of kyu. 13 kyu represent about two years of regular practise.

[9]World Computer Chess Championship.

[10]John McCarthy: a pioneer in Artificial Intelligence research. He was involved in the first international computer chess match U.S.A versus U.S.S.R. in 1966.

proverb summarizes this idea:

I hear and I forget

I see and I remember

I do and I understand

This is reinforced by the nature of the game itself in which a stone can't be moved once put on the board while in chess, moving one piece can completely change the face of the game. The GO knowledge, more than in any other game implies a **pattern recognition** ability that allows one to identify rapidly a situation and formulate the answer according to the **shapes** formed by the stones on the board. J. McCarthy suggested that *pattern recognition* combined with *local analysis* should lead to some breakthrough in Computer GO [11].

This strategy implies the existence of a database of GO problems used by the lessonware to make a lesson[12]. The choice of these problems depends on the criteria fixed by the student. IGOT can select a set of problems to make a lesson automatically according to an explicit representation of the student model. The parameters can be modified to produce a lesson on any specific tactical move, in any range of level, ignoring the student model[13]. We don't want IGOT to impose constraints that don't exist when the students read a book of GO problems. Curiosity is an excellent quality to exploit in assisting learning in a new domain and novices like to jump and read the last problems of the book, even if they can't solve them, just to get an idea of

[11]We don't mean that the process of analyzing a GO problem just requires an ability to identify typical shapes. It involves other analytic tasks as well.

[12]A lesson is a collection of GO problems

[13]This supposes that there are problems in the database that match the criteria.

what a hard problem is. IGoT gives this freedom to the students.

2.3.2 Transforming a problem into a game

The painstaking work of solving Go problems, for our style of tutor, meets three of our sub-objectives:

- To maintain a balance of the student's level of knowledge between the different categories.
- To raise the overall level of Go knowledge.
- To organize Go lessons.

IGoT does not reflect the behavior of the students in a game that is the other sub-objective. The reliable and fair method of the Go rating system (see Appendix B) encouraged us to consider each problem as a game. One player is the **computer** whose rating corresponds to the level of the problem[14] and the other is the student. The **handicap** is not known until the students give their answer after t seconds. Some information attached to the problem allows IGoT to compute the corresponding handicap. A rating is then attributed to the student and the problem is transformed into a game, called the **associated game.** While a game presents three possibilities (Win, Lose, Tie[15]), an answer can be graded from *mediocre* to *excellent* and thus permit the score of the associated game to be weighted accordingly.

[14]The notion of level of a problem is explained in section 2.5.

[15]There is no tie game in a match with handicap, because one half point is introduced in the counting.

The notion of *associated game* gives us a measure of the students' ability. The foundations of the rating system are solid and can be interpreted directly by a GO player. In a sense we achieve our initial goal of the ideal intelligent GO tutorial by dissecting a whole game into little games of a few stones each. All the stones played in a game correspond to the answers of new problems.

2.4 Overview of the System

The general architecture of IGOT is represented in Figure 2.1. We distinguish four components: a database of GO problems[16], a database of Student Models, the **lessonware**[17], and the interface. The first element is a collection of GO problems evaluated by the expert. All the GO knowledge of the system is gathered in that component. The student models contain all the information provided to IGOT concerning the students using the system. The *pedagogical organ* is the lessonware that has several tasks:

- To make decisions according to the student model and/or some parameters to **create a lesson** from the collection of GO problems.
- To **analyze** the answers to each problem of the lesson.
- To provide **explanation.**
- To **update** the student model.

[16]The notion of GO problems in the context of IGOT will be defined in section 2.5.

[17]Originally the *lessonware* like the *courseware* is the code for instructional applications distinguished from the computer code for general computer applications: the *software* [O'Neil 81].

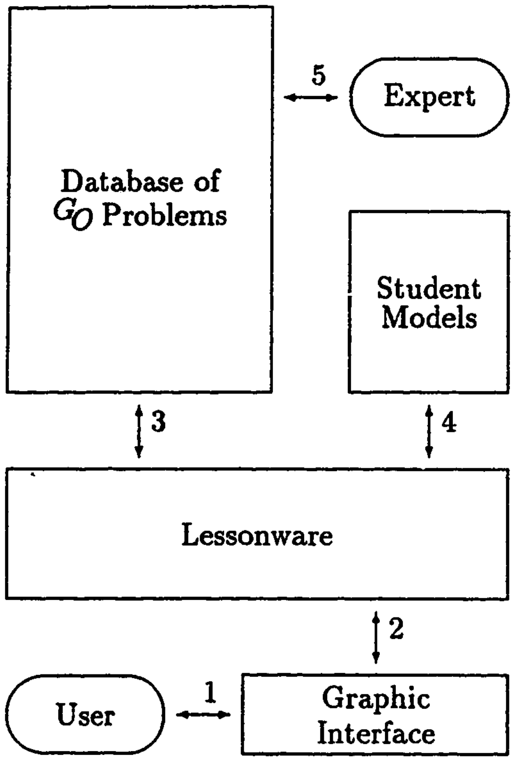


Figure 2.1: The general architecture of IGoT

The interface ensures the communication between the lessonware and the user with an appropriate language, the Go board representation being the most obvious. The relations between these elements are represented with arrows on Figure 2.1.

- **Link 1:** Mouse-driven communication. The interaction of the student with the system is described in Section 3.4.
- **Link 2:** Translation between the system's internal representation and the interface language.
- **Link 3:** Read access in both directions.

- **Link 4:** Write access for the lessonware.
- **Link 5:** The *Go* Expert is the only one to have access to the database of *Go* problems. The interaction of the expert with the *Go* library is explained in Section 3.1.3.

2.5 Reading and Intuition in *Go*

Reading

The notion of READING is the ability to evaluate the state of a problem or a game a few moves in advance by anticipating the opponent's answer and considering further moves. This is an analytical process, entirely equivalent to lookahead in computer chess ([Holding 86], pp 169–200).

Intuition

The notion of INTUITION represents the mental process of pattern retrieval. The game of *Go*, more than any other board game, involves an ability to recognize shapes. It is a non-analytic process. Eisenstadt and Kareev compared the effects of *Go* and Gomoku[18] knowledge on players' memory for patterns or stones that could have derived from either game ([Holding 86], page 114). They tested people with different levels of expertise in both games. The tests consisted of exposing game positions for

[18]Gomoku is game much simpler than *Go*, with many variants, played as *pegity* in England. The object, as in tic-tac-toe (noughts and crosses), is to complete an unbroken line of stones, pegs, or counters in any one direction.

a short time and have the players reconstructing these positions. The two knowledge systems appeared to be coded separately in the human mind, since providing *Go* players with Gomoku positions changed the form and the accuracy of the reconstruction they normally produced for *Go* positions. On the contrary, good Gomoku players proved to be much more accurate in reconstructing typical Gomoku positions. The difference in the reconstruction process was amplified at the master level. The unskilled players in both games seemed to have less well developed memory for patterns.

Reitman's experiment on pattern's retrieval in *Go* was done with a master and beginners, on real and random positions ([Holding 86], page 114). The *Go* master was substantially better than the beginners at memorizing the briefly exposed (5-seconds) game positions (66% correct versus 39%), whereas the skilled and unskilled players did not differ significantly when given random positions (30% versus 25%).

Go problem

In the particular context of this ITS a *Go* PROBLEM has two different parts that gather the ***explicit domain knowledge*** and the ***domain specific pedagogy.*** Figure 2.2 represents the content of a problem and we explain now the meaning c^f each item.

In the domain knowledge part, we just reproduce the basic information that can be found in a book of *Go* problems. The **board** is the list of the stones that form the problem. The **category** and the **concept** correspond to the chapter or the section of the book where the problem and the **question** are located.

The second part of the problem is specific to I*Go*T and the pedagogical strategy

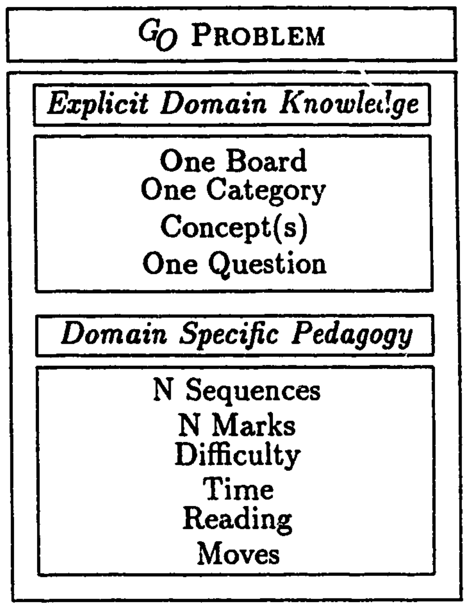


Figure 2.2: Definition of a Go Problem

involved ([Wenger 87], page 19). A problem can have different answers that are called **sequences.** A sequence is formed by one stone or a series of several stones when the first move doesn't give enough information on the understanding of the problem. Each of these sequences is graded by a **mark.** Then, as specified earlier, we associate a **level** to the problem that reflects the kind of player it is made for. The **reading** required to solve the problem is considered as well as the number of **moves** to think ahead to formulate a good answer. We define the Expert Module of IGoT as a collection of these Go problems.

2.6 The Descriptive Dynamic Student Model

2.6.1 The Units of Knowledge

The student models keep track of the students' aptitude provided by their reactions to the problems. We use the organization of the knowledge in the database of $\mathcal{G}_O$ problems to create an environment for representing the students' knowledge. The advantages are multiple:

- It reduces the problem of knowledge representation.
- A modification of the database of $\mathcal{G}_O$ problems, by adding new concepts, can be automatically reflected on the organization of the students' models, especially when building the database.
- The measurement concept is easier to manipulate because it is based on the $\mathcal{G}_O$ rating system.

The classification of $\mathcal{G}_O$ moves attached to the problems (see Appendix C) has lead us to the following definition:

The couple (CATEGORY, CONCEPT) is the basic unit of knowledge.

The student models contain a score for each of these *units of knowledge* that corresponds to the rating system. This implies that new users must have all their units of knowledge set to a default value equal to their official rating. Then the learning process modifies these scores and emphasizes the deficiencies.

The **average** of the units of knowledge represents the **rating** and the number of units of knowledge represents the **granularity** of our model. The actual system gathers about 100 units per student. Even if this seems too detailed, it is easy to manipulate and update because we are just concerned with numbers, ideal for mathematical and statistical operations.

In the TRAPS system[19], Peter Witschital uses the term **numerical tag** to indicate the *familiarity* of the student with a specific rule ([Witsc 89], page 599). There are minimum and maximum values for the familiarity. Tags are associated with relations indicating the *strength* of this type of relation. If students use a rule for the first time that is related to a known rule by a specific relation, it is assumed that they have discovered the new rule with the help of this relation. In this case the strength of the specific relation type is incremented. The relation with the highest strength value is the relation that is most used by the students in learning.

2.6.2 Mathematical Aspect of the Evaluation

Students get a number of points every time they do a problem. This score depends on the handicap and the result of the associated game. The notations are gathered in Table 2.1. The coefficient μ has a maximum of 1 for the best move. When the proposed answer doesn't match any of the expected moves, it is a wrong answer and the score of the student must be decreased by choosing an arbitrary value for $\mu = \mu_0$[20].

The difference of rating (see Appendix B) is given by the formula:

$$\Delta R = \Phi \times e^{\alpha\Theta}$$

[19]TRAPS is a gaming environment for novices in computer programming that want to gain first experience in using a structured programming language.

[20]We propose to begin with $\mu_0 = -0.5$

The symbols and their signification	
ΔR	The difference of rating
ΔK	The difference of knowledge
Θ	The handicap
α	The coefficient of probability
Φ	The coefficient of inertia
κ	Variable associated to a Category
σ	Variable representing a Concept
δ	Default value for the score attached to each couple (κ, σ)
μ	Evaluation of the answer

Table 2.1: The mathematical symbols.

We propose to transform the notion of **difference of rating** into a **difference of knowledge** as follows:

$$\Delta K = \mu \times \Delta R = \mu \times \Phi \times e^{\alpha\Theta}$$

This score ΔK enables updates to all the units of knowledge corresponding to the couples (κ, σ) that are attached to a $\mathcal{G}_O$ problem. Each score is updated every time a problem matches this area of knowledge by adding ΔK. These scores become better in some areas of knowledge and therefore can reflect the abilities of the students and their deficiencies. The lessonware can then select suitable problems for a well-balanced knowledge, leading to identical scores for each unit (κ, σ).

2.6.3 Inside the Student Model

Burton and Brown examined the computer-based coaching problem and all the subtleties that can be infered from observing the behavior of the student a little

Speed	Accuracy	
ρ_1	ρ_2	Reading
ρ_3	ρ_4	Intuition

Figure 2.3: The four features

deeper [Burton 82]. Similarly, the design of IGOT gives us different information on the attitude of the student that were not expected originally and their interpretation is somehow meaningful. It is possible to characterize the reading, the intuition, the speed and the accuracy of the students by analyzing their answers. We propose four factors to evaluate and measure the behavior. The four features, ρ_1, ρ_2, ρ_3 and ρ_4, presented in Figure 2.3 all have a different meaning. Their value should be about 0.5 to represent the norm. A factor of 1 means that the student has an excellent ability. Every answer to a problem modifies the four coefficients. For each of them we control the inertia of the modification with a weight η[21] and we have:

$$\rho^{new} = \eta \times \rho^{old} + (1 - \eta) \times \rho^{mod}$$

where ρ^{new} is the new value of ρ, ρ^{old} the value just before the last problem, and ρ^{mod} the factor of modification. The symbol T_0 represents the expected time for the student to solve the problem. For each problem, the lessonware knows how to evaluate T_0. In each of the functions given below, the norm corresponds to $T = T_0$.

- ρ_1 reflects the speed of the reading. We know that the reading speed is linked

[21] $\eta \in [0, 1]$ and it should be equal to about 0.9 to have a reasonable inertia on the problems solved.

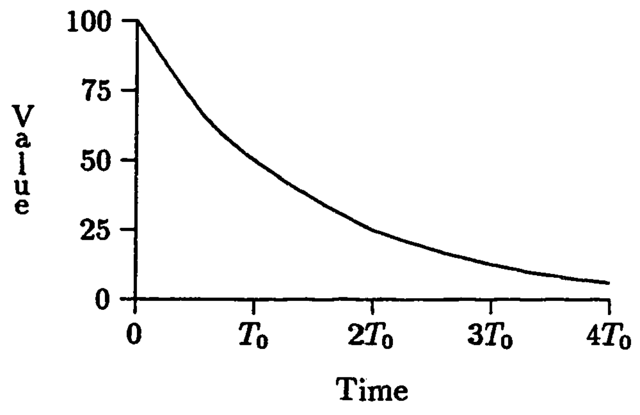


Figure 2.4: The factor ρ_1: reading speed.

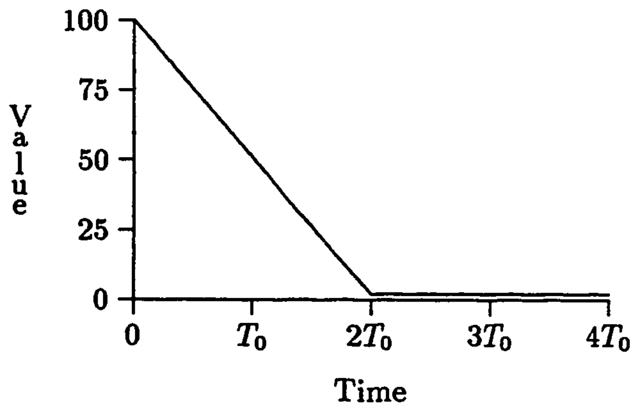


Figure 2.5: The factor ρ_3: intuition speed.

to the amount of extra time by the handicap required to solve the problem. If the student performs fast answers in problems such as *ladders*[22], this should be reflected in the score. (see Figure 2.4).

$$\rho_1^{mod} = e^{-ln(2)\times\frac{T}{T_0}}$$

- ρ_2 reflects the accuracy of the reading, the coefficient μ is the indicator for this

[22]A ladder is a simple concept that typically requires some reading ability.

feature.

$$\rho_2^{mod} = \mu$$

- ρ_3 is the speed of the intuition, that is to say the ability of the student to give an answer as soon as possible before T_0. If $T > T_0$ then there is no intuition at all because the students behave normally according to their strength. When $T \ll T_0$ the intuition is maximum. (see Figure 2.5).

$$\rho_3^{mod} = Max(1 - \frac{T}{2 \times T_0}, 0)$$

- ρ_4 reflects the ability of reading faster than expected and answering correctly.

$$\rho_4^{mod} = \rho_2^{mod} \times \rho_3^{mod} = \mu \times Max(1 - \frac{T}{2 \times T_0}, 0)$$

The four features are added to the student model as units of knowledge. Their value corresponds to a percentage instead of a rating.

2.7 Creating a Lesson Automatically

Let Γ be the set of categories and Ω the set of concepts. For each $\kappa \in \Omega$ and $\sigma \in \Gamma$ we note $\tau(\kappa, \sigma)$ the score related to the unit of knowledge (κ, σ). We need the following definitions of the average score in a category:

$$\forall \kappa \in \Omega, \Pi(\kappa) = \frac{1}{\|\Gamma\|} \times \sum_{\sigma \in \Gamma} \tau(\kappa, \sigma)$$

The algorithm presented in Table 2.2 creates a lesson of $\mathcal{G}_O$ in four steps. There are three degrees of liberty (**level, category** and **concepts**) affecting the knowledge

The steps for creating a lesson	
step 1	Define the range of **level**. The students can modify the level of the lesson even if it does not correspond to their rating.
step 2	Define the **category** κ_0 to work on by choosing the one corresponding to the lowest $\Pi(\kappa)$. Yet the students can impose this category if they want.
step 3	Define the **concept** σ_0 corresponding to the lowest $\tau(\kappa_0, \sigma)$. Here again the students are free to impose a set of concepts they want to work on.
step 4	Select randomly a set of **problems** from the database matching the unit (κ_0, σ_0). No influence of the students is allowed for this step.

Table 2.2: The creation of a lesson step by step.

provided in the lessons. Besides these degrees of liberty IGOT has some parameters like the maximum of number of problems per lesson. But they do not affect the quality of the problems selected, just their quantity. The quality of the lessons depends a lot on the quality of the database, and it should contain a reasonable amount of problems to work properly. This is the object of a discussion in Chapter 5.

Chapter 3

The Implementation of IGOT

3.1 The Tools

3.1.1 Hardware

IGOT is implemented on a compatible IBM-XT with a RAM of 640Kb that is part of a network supported by the Novell System[1]. We have chosen the IBM because *Kakari Systems*, Chuck Elliott's company, has developed a sophisticated environment to manipulate databases on IBM-PC. And the management of the database is crucial in IGOT. Moreover all the software necessary for the project was available. The graphics are designed for the EGA card that has a resolution of 640x350 and can display 16 different colors. A Microsoft-type mouse with three buttons is attached to the computer.

[1]Novell is a software system used to link IBM-PC's or compatibles into a local network.

3.1.2 Software

The code is written with the language **Clipper**[2] and the support of the **Flipper**[3] graphic routines library used for the interface. The *Flipper* package provides all the functions necessary for a Microsoft-type mouse[4]. This tool can directly indicate a precise location on the board. It helps us to create an efficient editor for Go problems and a user-friendly interface with pop-up windows as used on Macintosh[5]. *Clipper* allows the use of **DBase III plus**[6] commands to manipulate the database. The linker is **PLink86**.

3.1.3 The Go Library

We have written a program called **BOARD** to build a database of Go problems. This application is independent of the tutorial. It is a tool designed to facilitate the management of the expert module. The **BOARD** program is represented by the link 5 on Figure 2.1. This is the first attempt to create a Go library where the problems are evaluated and classified by an expert. There is a need for such a database in the Go world to help teachers in need of particular problems for a lesson. The library is flexible and allows one to retrieve and search problems. The expert enters the information of the problem in text mode. The stones on the board and the answers are recorded in graphic mode with the mouse. Many graphic routines developed for this editor are used in IGoT. In particular BOARD includes the **capture algorithm**

[2]Clipper Computer Summer 87, Copyright(c) Nantucket Corp 1985-1987. All rights reserved.
[3]Flipper 4.1. December 1988 copyright(c) 1988 ProWorks, All rights reserved.
[4]with three buttons to click.
[5]Macintosh is a product of Apple Computers.
[6]DBase III Plus is a product of Ashton Tate.

that identifies dead stones and removes them from the board.

The tutorial is built on top of this database, which means that if the structure of the database changes, the tutorial has to be modified. BOARD is just a tool to facilitate the management of the source of knowledge. All the problems used in IGoT must be recorded and evaluated by the expert. Anybody can enter new problems in the database, but a good knowledge is necessary to evaluate them. When some problems are overrated or underrated, it is always possible to modify the corresponding parameters without affecting the quality of the knowledge.

3.2 Internal Representation of the Go Board

The graphic interface must display Go boards with black and white stones and some particular symbols found in the Go literature. We have represented in Table 3.3 most of the symbols of the new font **GO1** created with *Flipper*. The internal encoding of the Go board and its translation with the font *GO1* are represented in Figure 3.6. The advantage is that manipulating stones on the board is equivalent to modifying strings of characters and *Clipper* has special functions for this purpose.

3.3 Database Architecture

3.3.1 General Organization

The database is implemented in the **DBase III plus** environment. Different files have been created to save space and allow the lessonware an easy access to the information. They are listed in Table 3.4. The database has two poles, as represented

○	a	┤	j
●	b	┴	k
┼	c	╋	l
┌	d	a	m
┐	e	b	n
└	f	c	o
┘	g	d	p
├	h	◬	CHR(91)
┬	i	▲	CHR(92)

Table 3.3: The symbols of the font GO1.

in Figure 3.7: **the problem** holds the information concerning the *Go* problem and **the player** gathers all of what is known about the student. The role of the files in the database divides them into two categories:

The passive files that are basically static descriptive files that contain some text attached to an index. For instance the concept file ensures the correspondence between concept numbers and their meaning in English. They allow us a better maintenance of the database, an easier manipulation with the indexes and they save space. In the current version only the human expert can modify them.

The active files are crucial in the architecture of the database. They are the important organs that gather the minimum data necessary to manipulate the information. The content of some of them can be modified by the lessonware. Arteries link these organs as represented in Figure 3.7 and the heart of the system is the *CAT_CON* file that ensures the same units of knowledge for both poles of the database.

Name	Type	Description
BTYPE	Passive	Types of boards available.
CAT	Passive	Description of the different categories.
CAT_CON	Active	List of the the couples (Category Concept).
CONCEPT	Passive	Description of the different concepts.
MESSAGE	Passive	List of the available messages associated to a sequence.
PLAYER	Active	General information on the players.
PLAY_CON	Active	Representation of the Student Model.
PROBLEM	Active	General information concerning the *Go* problems.
PROB_LNK	Active	Link Problem, Category, Concepts.
REF	Passive	List of the Books from where the problems are taken.
SEQ	Active	List of the different answers to the problems.

Table 3.4: List of the files in the database

Section 3.3.2 and Section 3.3.3 discuss in more detail the content of these files and their relations.

3.3.2 The Files Attached to *Go* Problems

Anders Kierulf proposed a Standard File Format for *Go* Games [Kierulf 87a]. This is of great help in exchanging games and building databases to provide *Go* services. We cannot match the requirements of IGoT with the format proposed because it is designed for games instead of problems. Therefore many features of the standard format remained unused while some options necessary for the recording of the pedagogical part where unavailable. A rapid access to the information contained in the problem implied the use of a database protocol like *DBase III plus* instead of a simple text file. Nevertheless we have adopted the system of coordinates presented in the paper.

Three active files characterize the GO problems. The starting point is the PROBLEM file (see Table 3.5) that matches the definition of a GO problem given earlier. It is divided into two parts. All the *fields*[7] represent explicit GO knowledge except *MOVES, DIFF, SECONDS, READING* that correspond to the specific pedagogy adopted by IGOT and can only be fulfilled by the human expert. The levels of difficulty go from 0 to 9, and each of them corresponds to a range of ratings as defined in Table 3.6.

The structure of the memo for the *STONES* field allows *DBase III plus* to expand automatically its length by blocks of 127 bytes when one is not enough. This field is a string of characters that represent the symbols to be drawn on the board. These symbols can be black or white stones of special symbols found in the GO literature and present in GO1. Each takes three characters of the string *STONES*.

(Symbol)(X-position)(Y-position)

The position of a stone is represented with small letters and (a,a) corresponds to the upper left position on the partial board[8]. Figure 3.8 represents the board of a GO problem with its characteristics. The types of partial board used are kept in the file BTYPE (see Table 3.7).

[7] A field is any item of a file in the database.

[8] We call partial board the part of the board represented in a problem (e.g. the lower part, the left part, the upper right corner, the full board).

The string *STONES* presents an implicit order among the symbols. In our example, Figure 3.8 the stone (Aac) is the first stone and the last one is (Bcc). This feature allows us the recording of entire games in which the order of the stones played and the prisoners (see Appendix A) are crucial, without space limitation[9]. The field *BTYPE* distinguishes the stones that belong to:

A problem with multiple choice in which the number of answers is limited to a few intersections clearly indicated on the board.

A problem with sequences where there are several possible answers, some better than others, and composed of a series of moves or **sequences.**

A game where the order of the stones is more important than their symbols and where the prisoners have to be taken into account.

PROB_LNK (see Table 3.8) attaches units of knowledge to a *Go* problem that always belongs to only one category, but often covers multiple concepts. All these units of knowledge must belong to the CAT_CON file (see Table 3.9) that is the heart of the system. The CAT (see Table 3.10) and CONCEPT (see Table 3.11) files contain the significance of the indexes of categories and concepts that are stored in PROB_LNK and CAT_CON. The references of the problems are stored in the file REF (see Table 3.12).

The third active file for the *Go* Problem is the SEQ file (see Table 3.13) that contains the answers to the problems. Whether it is multiple choice or not, there are several answers that are numbered in the field *SEQ*. The field *MOVES* contains the

[9]The memo *STONES* is expandable.

sequences of moves forming the answer. The order of the stones is important here as in the recording of a game. The field *NEXT* in the file PROBLEM determines the color of the first stone; then it alternates between black and white. Nevertheless, we maintain the format of three characters per symbol that does not considerably increase the space for most of the sequences are made of a few stones, which is negligible compared to the size of the problem, and for which the graphic routines designed for *STONES* work fine. *MESSAGE* is a brief comment associated with the sequence (*e.g. best choice, excellent, wrong answer, weak move*). The messages are stored in the MESSAGE files (see Table 3.14). To evaluate each sequence, the expert puts a mark ranging from 0 (bad) to 9 (excellent).

3.3.3 The Files for the Student Modeling

The pole of the players is composed of two elements: the PLAYER file (see Table 3.15) and the PLAY_CON file (see Table 3.16). General information is stored in the first one while the second constitutes the real student model. It contains the scores associated to each of the units of knowledge present in the CAT_CON file plus the four features introduced in Section 2.6.3.

```
diiiiiiiiiiiiiiiiie
hcccccccccccccccccj
hcccccccccccccccccj
hcclccccclccccclccj
hcccccccccccccccccj
hcccccccccccccccccj
hccccccccmccccccccj
hcccccccccccccccccj
hcccccccccccccnccccj
hcclccccclccccclccj
hcccccccccccccccccj
hcccccccbcccccccccj
hcccccacccccccccccj
hcccccccccccccccccj
hcccccccccccccccccj
hcclccccclccccclccj
hcccccccccccccccccj
hcccccccccccccccccj
fkkkkkkkkkkkkkkkkkg
```

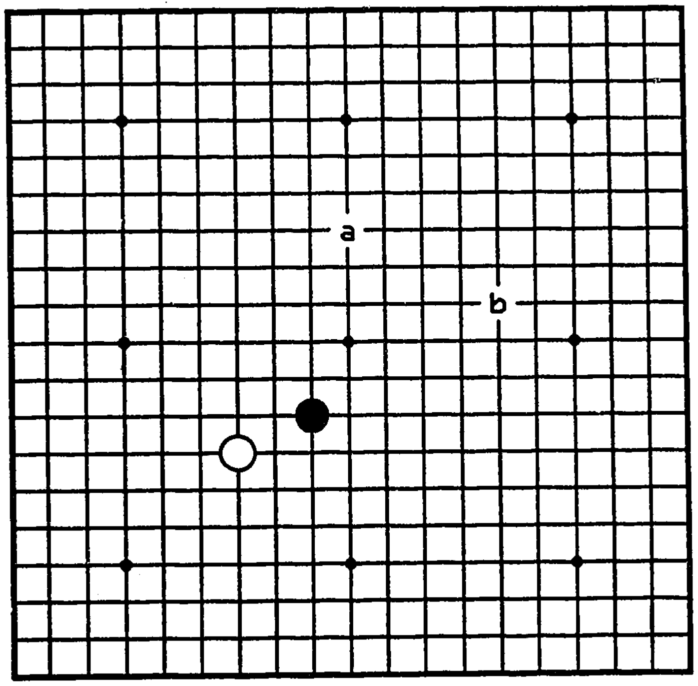


Figure 3.6: The *Go* board representations.

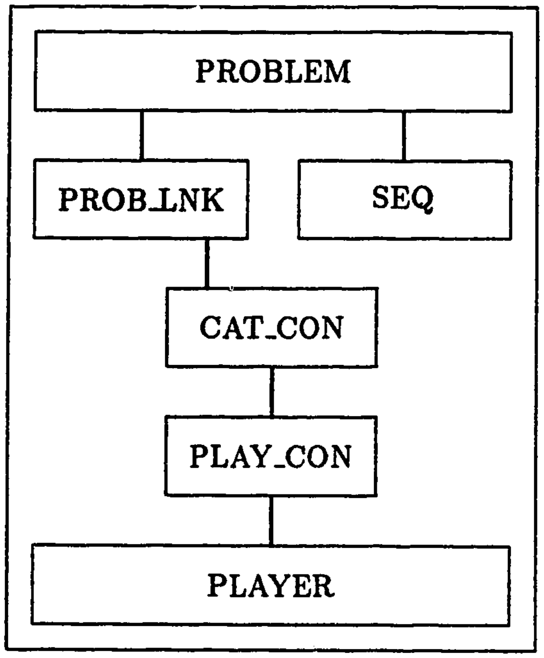
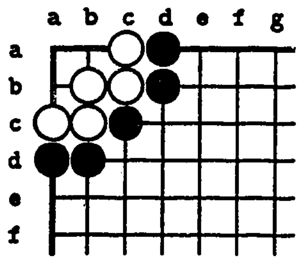


Figure 3.7: The architecture of the database



```
BTYPE    : UL  (Upper Left)
HEIGHT   : 6
WIDTH    : 7
STONES   : <AacAbbAbcBadBbdAcaBdaBdbAcbBcc>
```

Figure 3.8: Example of *Go* problem representation

PROBLEM			
	Structure		
Field	*Type*	*Length*	*Comments*
ID	N	5	Index of the *Go* problem.
REF	A	6	Origin of the problem.
PAGE	N	3	Page.
DIAGRAM	N	3	Number of the diagram.
BTYPE	A	2	Type of board.
HEIGHT	N	2	Height of the partial board.
WIDTH	N	2	Width of the partial board.
NEXT	A	1	Next to play (B or W).
MOVES	N	2	Number of moves for the answer.
DIFF	N	1	Level of difficulty.
SECONDS	N	4	Number of seconds to solve the problem.
READING	A	1	Indication of the reading required.
CAT	A	3	Index of the category attached to the problem.
STONES	Memo	127x	Stones forming the *Go* board.
COM1	A	30	Question for the student.
COM2	A	30	Question for the student.
COM3	A	30	Question for the student.
COM4	A	30	Question for the student.
COM5	A	30	Question for the student.
COM6	A	30	Question for the student.
PTYPE	A	1	Type of *Go* Problem (Multiple choice (M), Sequence (S) or Game (G)).

Table 3.5: Structure of the PROBLEM file

Problem Level	
Level	Rating
0	30-19 kyu
1	18-16 kyu
2	15-12 kyu
3	11-8 kyu
4	7-4 kyu
5	3-2 kyu
6	1 kyu-1 dan
7	2-3 dan
8	4-5 dan
9	6 dan and up

Table 3.6: The different levels of difficulty in IGoT

BTYPE			
	Structure		
Field	*Type*	*Length*	*Comments*
BTYPE	A	2	Index that identifies a type of partial board.
DESC	A	15	String of characters attached to the index (Upper Left, Lower right, Full, Left, etc...).

Table 3.7: Structure of the BTYPE file

PROB_LNK			
	Structure		
Field	*Type*	*Length*	*Comments*
ID	N	5	Index that identifies the Go Problem.
CAT	A	3	Index that identifies the category.
CONCEPT	N	3	Index that identifies the concept.

Table 3.8: Structure of the PROB_LNK file

CAT_CON			
	Structure		
Field	*Type*	*Length*	*Comments*
CAT	A	3	Index that identifies a type of board.
CONCEPT	N	3	Index that identifies the concept.
CAT_DESC	A	12	Description attached to the category.
CON_DESC	A	30	Description attached to the concept.

Table 3.9: Structure of the CAT_CON file

CAT			
	Structure		
Field	*Type*	*Length*	*Comments*
CAT	A	3	Index that identifies a category.
DESC	A	12	String of characters attached to the category.

Table 3.10: Structure of the CAT file

CONCEPT			
	Structure		
Field	*Type*	*Length*	*Comments*
CONCEPT	N	3	Index that identifies a concept.
SHORT	A	10	Short description of the concept.
DESC	A	40	Long description of the concept.

Table 3.11: Structure of the CONCEPT file

REF			
	Structure		
Field	*Type*	*Length*	*Comments*
REF	A	8	Index that identifies the origin of the problem.
TITLE	A	50	Title of the book where the problem can be found or any other reference.

Table 3.12: Structure of the REF file

SEQ			
	Structure		
Field	*Type*	*Length*	*Comments*
ID	N	5	Index that identifies a problem.
SEQ	N	2	Number of the sequence.
MOVES	A	100	Stones associated to the sequence.
MESSAGE	N	2	Index of the message associated to the sequence.
GRADE	N	1	Mark associated to the sequence.

Table 3.13: Structure of the SEQ file

MESSAGE			
	Structure		
Field	*Type*	*Length*	*Comments*
MESSAGE	N	2	Index that identifies a message attached to a sequence.
DESC1	A	50	First part of the message attached to the index.
DESC2	A	50	Second part of the message.

Table 3.14: Structure of the MESSAGE file

PLAYER			
	Structure		
Field	*Type*	*Length*	*Comments*
P_ID	N	4	Index that identifies a player.
NAME	A	20	Name of the student.
RANK	N	2	Score in the rating system (15 for 15 kyu).
CLASS	A	3	In the rating system, difference between *dan* and *kyu*.
VAR1	N	2	ρ_1.
VAR2	N	2	ρ_2.
VAR3	N	2	ρ_3.
VAR4	N	2	ρ_4.

Table 3.15: Structure of the PLAYER file

PLAY_CON			
	Structure		
Field	*Type*	*Length*	*Comments*
P_ID	N	4	Index that identifies a player.
CAT	A	3	Category of the unit of knowledge.
CONCEPT	N	3	Concept of the unit of knowledge.
SCORE	N	5	Score attached to the unit of knowledge.

Table 3.16: Structure of the PLAY_CON file

3.4 The Interface

3.4.1 Presentation

The means of communication between IGOT and the student are the screen and the mouse. While the reactions of the system are expressed with a graphic language, the students' reactions are transmitted with the mouse position and the clicked button. These constitute the only tool the students have to interact with the system. The interface is highly graphical and makes extensive use of menus and mouse-sensitive graphical objects as in Spada's DiBi system, a computerized learning environment for elastic impacts [Spada 89]. The screen is divided into six regions as presented in Figure 3.9:

- A **status region** that keeps track of the customized lesson and the general variables.
- A **question region** where a few lines indicate what the problem is.
- A **time region** where the clock simulates the time passing according to the expected time for the strength of the student.
- A **comment region** where some brief indications of the moves is given.
- A **board region** where the board and the stones are displayed and where the answer is completed by clicking an intersection.
- A **menu region** where clicking pop-up sub-menus windows as represented in the Table 3.17.

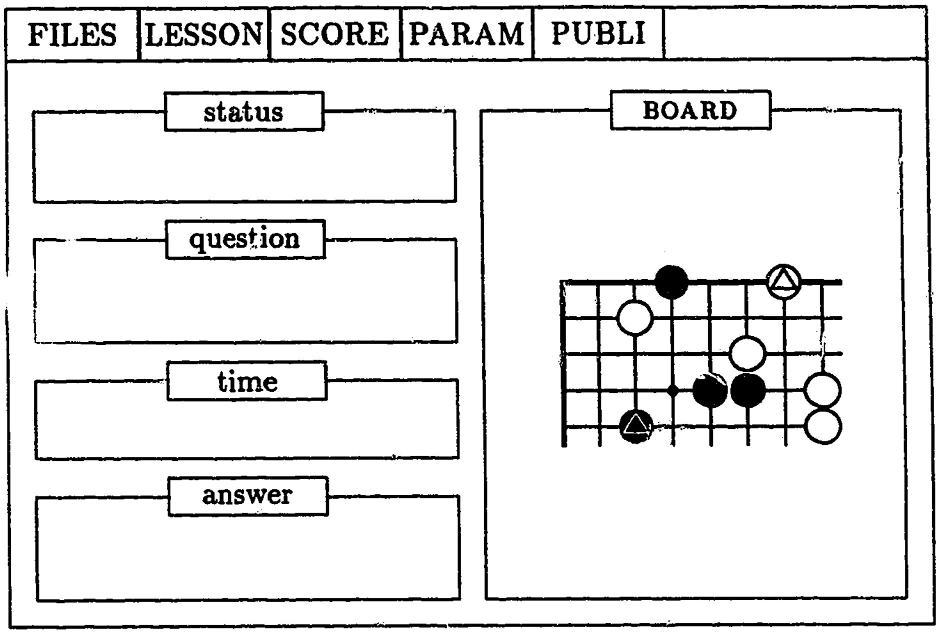


Figure 3.9: The aspect of the screen in IGoT.

3.4.2 Overall Behavior of the System

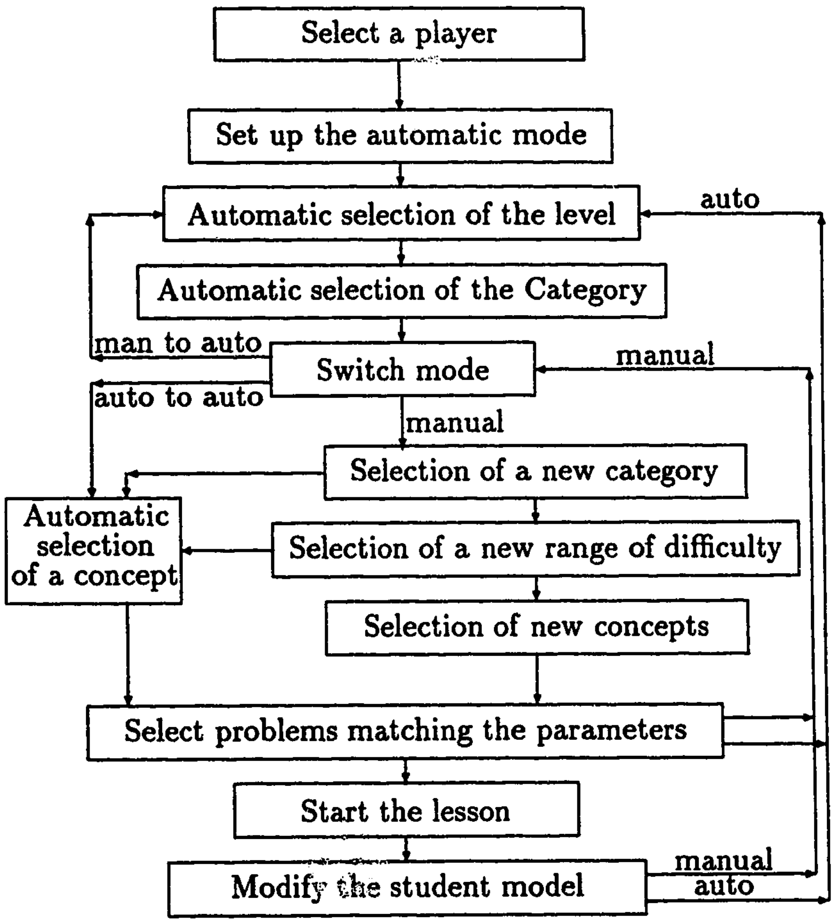
Upon presentation with the graphical interface (Figure 3.9) students interact with the system, by selecting a set of problems that form a lesson. These problems must match various criteria (category, range of level, concepts) on which the students can have a total control if they wish. The flowchart in Figure 3.10 represents the different steps of the dialogue between the tutor and the student and illustrates the different degrees of liberty left to the student in the creation of a lesson. Even in manual mode, the student has great flexibility since the concept can be chosen automatically by the tutorial system. The problems are chosen randomly by the system from the library, according to the parameters of the lesson. One can notice that selecting a

FILES	LESSON	SCORE	PARAM	PUBLI
quit	New	Pie	Category	Lesson
	Close	Chart	Concepts	One Prob
	Start	Special	Level	Scores/cat
			Manu/Auto	Model
			max Prob	

Table 3.17: The windows of the submenus in I$^{\text{G}}_{\text{O}}$T.

set of problems doesn't imply that a lesson has to start. For instance a teacher can use the tutorial to select a set of problems in a specific area of knowledge and have them printed to provide some handouts during a lecture. In that case taking the lesson with the system is not useful.

Students can also interact with the system between the lessons. There are some options to display the different features of the student model under the SCORE menu. They inform the students on their performances, the shortcomings in their knowledge and give them an global estimation of their rating.



Select a player
Set up the automatic mode
Automatic selection of the level
auto
Automatic selection of the Category
man to auto
Switch mode
manual
auto to auto
manual
Selection of a new category
Automatic selection of a concept
Selection of a new range of difficulty
Selection of new concepts
Select problems matching the parameters
Start the lesson
Modify the student model
manual
auto

Figure 3.10: High-level operations of the system.

3.4.3 Customizing a Lesson

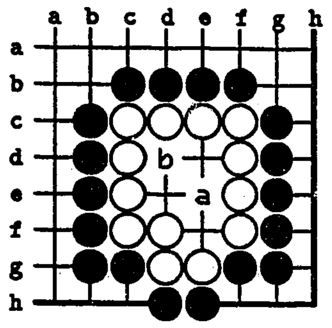
The menu LESSON allows one to switch from the *automatic mode*, in which the lesson is entirely created by the lessonware, according to the student model, to the *manual mode* where the parameters can be modified by the student. These parameters, displayed in the STATUS region, are the range of difficulty, the maximum number of problems, the category and the subset of concepts. The PARAM menu can be clicked to display a pop-up window (see Table 3.17). A scrolling window is brought up by clicking an option on the screen where the users can browse through the different possibilities and select them. The parameters conform to several rules:

rule 1: The concepts have to be chosen among the ones attached to the category selected. To select other concepts, the user has to choose another category they belong to first. Hence if two concepts are never in the same category it is impossible to select them together.

rule 2: At any time, *minimum level*[10] $\leq$ *maximum level.*

rule 3: The *Manual* option is toggled to *Auto* when clicked. The role of the *Auto* function is to default all of the parameters according to the representation of the student. To modify the parameters of the lesson, *Manual* mode must be on.

[10]Level of difficulty attached to the problems.



```
Black to play.
What is the vital point for White ?
Is it a or b ?
```

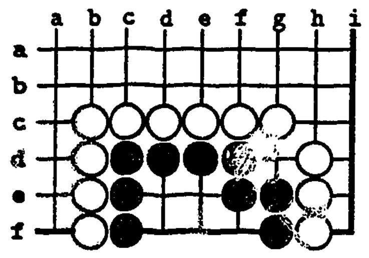
Figure 3.11: Example of a $\mathcal{G}o$ problem with multiple choice

3.4.4 Displaying a Problem

A problem in a lesson is composed of a board and a question. There are two kinds of problems to display. The first is represented in Figure 3.11 and corresponds to a problem with multiple choices. The second, Figure 3.12, is a problem with sequences.

When the problem and the question are displayed on the screen, a timer appears in the time region. It is represented in Figure 3.13. The first rectangle is filled progressively so that the end corresponds to the expected time. The second is filled with blocks, one at a time, whenever the first rectangle is full. A total of five blocks is allowed and the first rectangle is cleared every time a new block is added. The limitation of five blocks is arbitrary but avoids inconsistency resulting from infinite times giving the answer.

The expected time depends on the nature of the problem and the knowledge of the student. IGoT estimates the rating of the student as the average of the scores in



```
White to play.
Where to play to kill the black group of stones ?
```

Figure 3.12: Example of GO problem with sequences

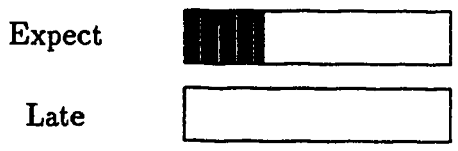


Figure 3.13: The representation of the timer in IGOT.

the units of knowledge. Then it assumes that the time required to solve a problem is a linear function of the rating as represented in Figure 3.14. The experience (see Chapter 4) proves that simplification is close to reality, at least for a reasonable range of levels. The PROBLEM file gives us one point of this curve, if we decide that the field *TIME* corresponds to the highest rating for the *LEVEL*. For instance, level 1 is associated to 16 kyu according to the table of levels. The difficulty is to evaluate the slope. According to our tests (see Chapter 4) we propose the following formula:

$$slope = -\frac{(R+1) \times LEVEL \times MOVES}{100}$$

where R is the reading. Its value is 1 when the problem requires reading, 0 otherwise. For a problem of level 3 requiring reading ($R = 1$) and one move, the slope equals

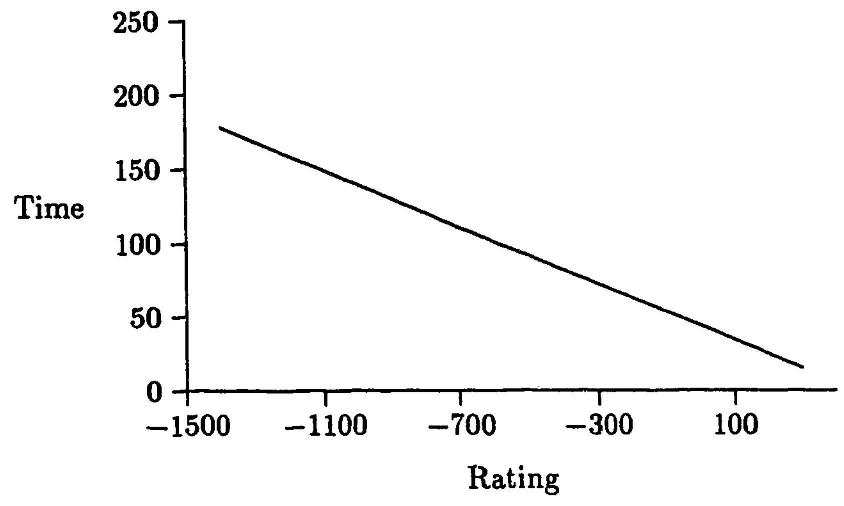


Figure 3.14: The linear relation between expertise and time for solution.

-0.06 which matches the results of the test. This slope (its absolute value) should increase with the level of the problem and the number of moves required for the answer. Once we have the slope of the curve, all the points can be calculated, and the expected time evaluated.

3.4.5 Analyzing Answers

When the user clicks one intersection on the board different scenarios are possible. First there are two rules for a move to be valid.

Rule 1: A problem with multiple choice allows only these choices (a,b,..) and nothing else.

Rule 2: The move has to be valid regarding the rules of the game of *Go* (no suicide).

Once the first move is valid the timer stops. In case of a problem with multiple choice the answer is straightforward after the first stone. A message is displayed

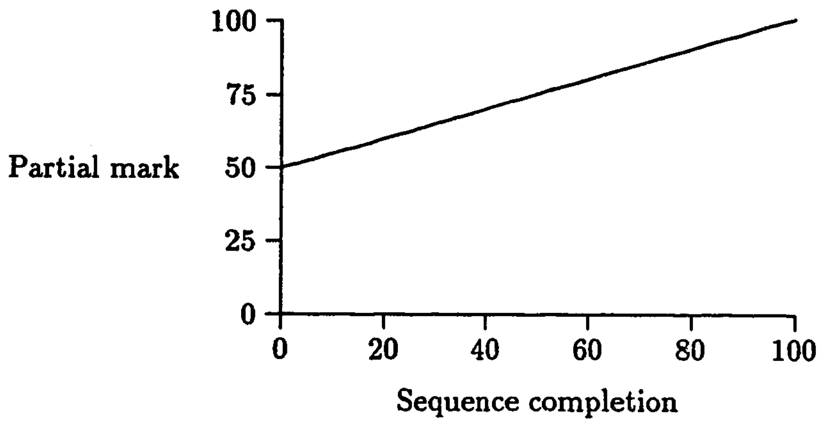


Figure 3.15: Partial mark for long sequences.

in the answer window (e.g. best move, second best move). When the choice is not the best, the user must press the keyboard to display the next move. Another *Go* problem appears when the keyboard is pressed anew.

For a problem with sequences, IGoT tries to match the first move to the best answer in decreasing order. As long as the move is not the last of the sequence, the computer plays every other move of the sequence. The process continues until the user plays a move that does not match any sequence. The answer window shows a message and the right answer is displayed by pressing any key. The further the users go in a sequence, the higher their mark is. The maximum they can get is the mark attached to the sequence. For the first correct stone they get half of the mark, then the partial mark is a linear function of the percentage of the sequence completed as shown in Figure 3.15. The prisoners are removed when they are captured.

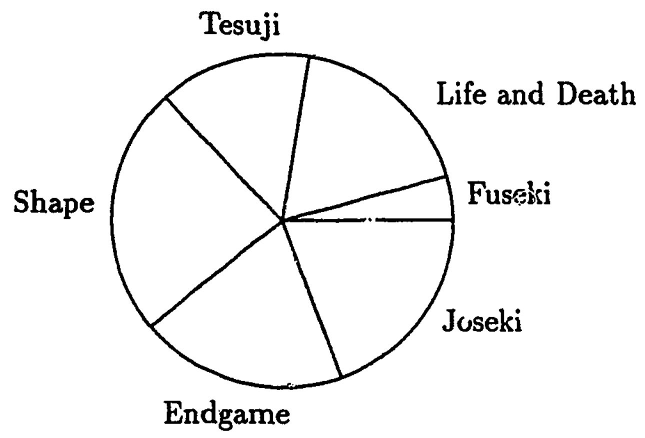


Figure 3.16: The distribution of *Go* knowledge between the categories

3.4.6 Displaying the Student Model

The score menu helps one to visualize the state of the student model. The pie option displays the distribution of *Go* knowledge between the different categories with percentages (see Figure 3.16). The smallest slice of the pie corresponds to the choice of I*Go*T in the automatic mode to organize a lesson.

The chart option provides a diagram with bar charts for each category and the average. This allows one to compare the amount of knowledge in each of them with the average rating (Figure 3.17). Actually the scores represented are the difference between the score in the student model and the value -2000 that is the lowest value for a score. In the example on Figure 3.16, the average is 665 and it means that the estimated rating of the user is -1335, that's to say 14 kyu. The knowledge is not properly balanced. The knowledge in *Fuseki* is better than the ability in *Life and Death* problems. The other categories are closer to the average and the norm of the model.

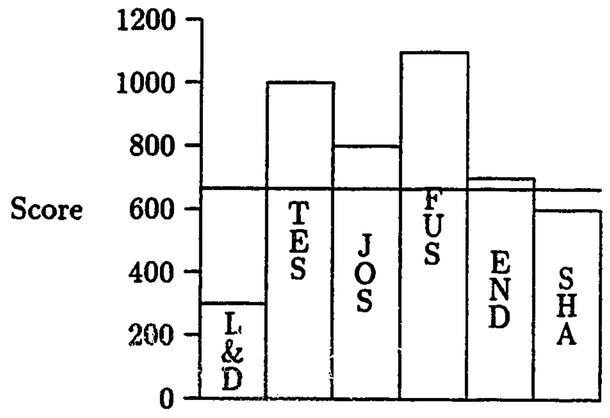


Figure 3.17: The knowledge in each category compared to the average

Chapter 4

Tests and Results

4.1 Presentation of the Test

The previous chapters explain how we built I^{G}$_{O}$T. We did not give any strong justification of the underlying theory that conveys some intelligent features to the system. Our assumption was based on the pedagogical experience of the expert. The object of this chapter is to prove the narrow relation between the expertise and the time spent on a problem. We made the assumption that time is a linear function of the strength. The test we prepare has to prove that this assumption is reasonable and close to reality.

We select statistically significant random problems from the *Go* literature. These problems are taken from the four sections of the book and scan most of the categories we described earlier. For each of them we record the time required to give the answer, marked *Wrong* or *Right*. The players must complete all the problems, one after the other, as quickly as possible. This procedure has the advantage of isolating each problem and providing the total amount of time spent on it. The conditions are the same as in IGoT that displays one problem at a time.

The choice of the players and the problems has an obvious impact on the results. We should not expect from this test rigorous proofs of our assumptions. First the rating communicated by the players is not always accurate and can vary depending on the club they belong to and the frequency of their games. Two players with the same strength can have a different knowledge of Go. One may be excellent in the Endgame phase while the other excels in Life and Death Problems. On one given problem, the two players will show a different behavior even if their rating is the same. Obviously 12 problems don't encompass all the Go knowledge. Nevertheless the sample should bring some interesting information forward.

4.2 Results

The results of the tests are displayed in Table 4.19. Nine players have been tested. A quick look at the results shows that the number of correct answers varies from one player to the other. Some take their time and have many correct answers. Others rush resulting in many mistakes. For weak players that are lost after a few minutes, what does the time mean? We must reflect somehow the behavior of the students during the test to have a more accurate perception from the results. Figure 4.18 represents the total time spent on the test for each player, while Figure 4.19 shows the relation between the rating and the number of correct answers. The ratings of the players range from 14 kyu to 3 dan. The time is given in seconds.

Prob	*r*	*b*
Pb 88	-0.83	-0.09
Pb 94	-0.89	-0.08
Pb 100	-0.97	-0.08
Pb 137	-0.99	-0.04
Pb 148	-0.79	-0.08
Pb 142	-0.86	-0.07
Pb 147	-0.94	-0.14
Pb 138	-0.96	-0.03
Pb 154	-0.67	-0.03
Pb 167	-0.84	-0.20
Pb 168	-0.81	-0.13
Pb 169	-0.72	-0.04
Average	-0.86	-0.08

Table 4.18: The Variance and the Regression Coefficient.

For each problem we propose a linear regression. The correlation[1] coefficient is given by the formula:

$$r = \frac{\sum xy}{\sqrt{\sum x^2 \sum y^2}}$$

where[2] $x = X - \overline{x}$ and $y = Y - \overline{y}$. Its value varies between -1 and +1 and represents the accuracy of the linear approximation. A value close to 0 indicates a bad linear approximation. We indicate it in Table 4.18.

It appears that the results obtained for the relation between time and rating are not regular. We decide to select five players who cover a wide range of ratings and who have a good correlation to do statistics on this sample. The players are the number 2, 4, 5, 7 and 9. The values of the correlation in the relation between rating

[1]Correlation, like covariance is a measure of the degree to which variables vary together or a measure of the intensity of association ([Steel 60], pp 183–185)

[2]x represents the distance of the variable X to the average $\overline{x}$

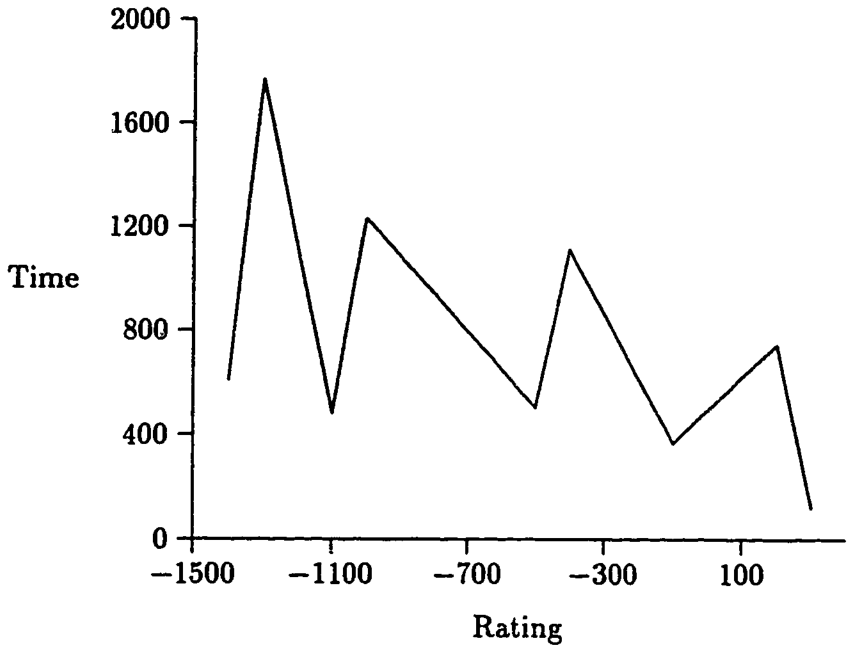


Figure 4.18: Relation between *Rating* and *Time*

and time is given for each problem in Table 4.18. The average *correlation factor* of -0.86 on the twelve problems illustrates the linear behavior of the candidates. The experiment should be repeated with more people to show better correlations. Each problem can not reasonably be applied to a range of rating such as the one chosen here. It is certainly better to prepare sets of problem in a narrower range of difficulty and test them with players situated in these ranges of levels. The idea is that the linear correlation is probably the highest in the neighborhood of the level rather than on the complete range of *Go* players. This feature matches the design of IGoT. A level is associated to each problem and used for lessons where the range of difficulty covers it. If this range is narrow, our assumption is correct.

A complete study of the linear relation between time and expertise would take a

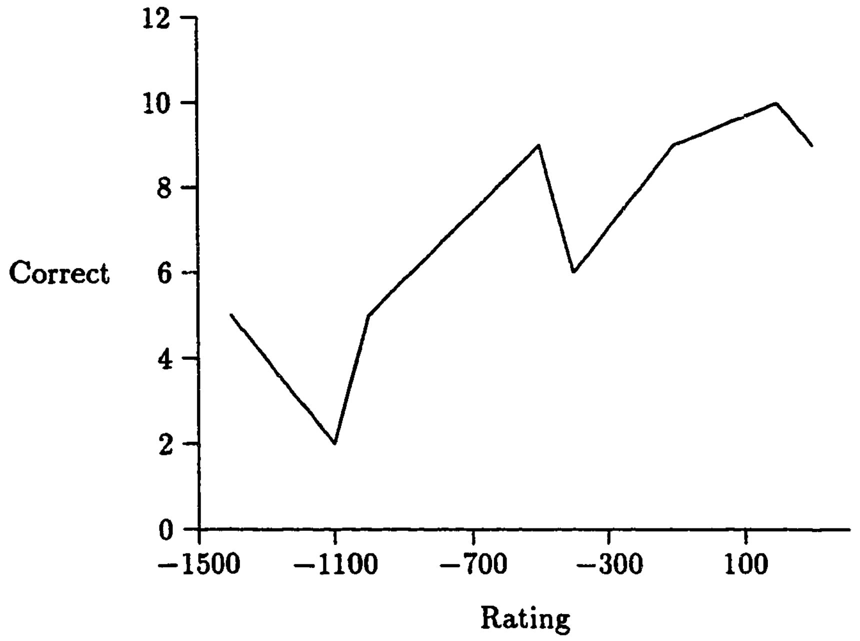


Figure 4.19: Relation between *Rating* and *Correct Answers*

lot of effort and might not even work. There are different alternatives for conducting the test. It can be done in a fixed amount of time during which the players have to solve different problems. The sample of players tested should model the distribution of *Go* players, reflecting the few good players available compared to the numerous intermediate players. The problems may reflect the variability one can expect between two players of the same level of expertise. It would be important to further investigate the time/expertise relationship. But for the need of the current tutorial the linear assumption is good enough. it doesn't affect the quality of the knowledge in the database anyhow, only some students may be overrated or underrated with certain problems.

4.3 The Slope of the Time-Strength function.

Our tests allow us to find the value of the slope in the diagram Time = F (Rating). The value of the slope is calculated with a linear regression. We recall here the formula of the *regression coefficient* [Steel 60]:

$$b = \frac{\sum xy}{\sum x^2}$$

where x and y have the same definition as above. In Table 4.18 we show the value of the regression coefficient for the weighted case that is the most significant. For the range of problems (level 2-4), the variations of the slope illustrate the formula introduced in the previous chapter:

$$slope = -\frac{(R+1) \times LEVEL \times MOVES}{100}$$

The problems 94, 100, 148, 142 require about two moves (one move is not enough), some reading, and their level is about 2. The formula gives:

$$b = -\frac{(R+1) \times LEVEL \times MOVES}{100} = -\frac{(1+1) \times 2 \times 2}{100} = -0.08$$

The problem 167 (b=-0.20) is estimated of level 3 and it requires reading and an ability to read at least three moves in advance: the formula gives a slope of

$$b = -\frac{(R+1) \times LEVEL \times MOVES}{100} = -\frac{(1+1) \times 3 \times 3}{100} = -0.18$$

In the problems 137, 138 and 154 intuition is required instead of reading:

$$b = -\frac{(R+1) \times LEVEL \times MOVES}{100} = -\frac{(0+1) \times 3 \times 1}{100} = -0.03$$

These values are arbitrary but they approach the reality. We could imagine adding a new field in the $\mathcal{G}o$ problem file that would be called SLOPE and contain the value b. This value could then be corrected and updated with some feedback from the observation of the student's behavior (Chapter 5).

Number	*Player1*		*Player2*		*Player3*		*Player4*		*Player5*	
Name	**Lin**		**Sam**		**Kevin**		**Michael**		**Peter**	
Rating	*14 Kyu*		*13 Kyu*		*11 Kyu*		*10 Kyu*		*5 Kyu*	
Prob	*Ti*	*An*	*Ti*	*An*	*Ti*	*An*	*Ti*	*An*	*Ti*	*An*
Pb 88	39	W	178	R	73	W	84	W	5	W
Pb 94	78	W	110	W	60	W	139	W	29	W
Pb 100	34	W	116	W	16	R	110	R	41	R
Pb 137	50	W	76	R	6	W	60	R	47	R
Pb 148	54	W	90	W	80	R	174	W	55	R
Pb 142	50	W	93	R	17	W	136	W	25	R
Pb 147	69	R	265	W	50	W	160	R	87	R
Pb 138	51	R	51	R	19	W	52	R	24	R
Pb 154	32	R	90	R	35	W	24	W	54	R
Pb 167	70	W	395	W	75	W	130	R	59	R
Pb 168	46	R	263	W	23	W	66	W	35	R
Pb 169	36	R	39	W	28	W	95	W	40	W
Total	609	5	1766	4	482	2	1230	5	501	9

Number	*Player6*		*Player7*		*Player8*		*Player9*	
Name	**Ron**		**Fred**		**Wlodek**		**Mark**	
Rating	*4 Kyu*		*1 Kyu*		*2 Dan*		*3 Dan*	
Prob	*Ti*	*An*	*Ti*	*An*	*Ti*	*An*	*Ti*	*An*
Pb 88	55	W	31	R	20	R	15	R
Pb 94	37	R	26	W	65	R	9	W
Pb 100	173	R	24	R	32	W	3	R
Pb 137	112	W	23	W	69	R	8	R
Pb 148	83	W	29	W	122	R	10	R
Pb 142	84	W	25	R	80	R	1	R
Pb 147	50	R	53	R	67	R	35	W
Pb 138	19	W	22	R	69	W	12	W
Pb 154	45	R	41	R	77	R	14	R
Pb 167	288	R	45	R	45	R	14	R
Pb 168	82	W	25	R	37	R	6	R
Pb 169	80	R	21	R	60	R	2	R
Total	1108	6	365	9	743	10	120	9

Table 4.19: The Results of the Test.

Chapter 5

Limits and Improvements

5.1 The Database

5.1.1 The Content of the Database

The lessonware depends on the source of knowledge. If the problems available are poorly chosen the tutorial system is useless. Therefore the knowledge must match the features of the tutorial system and make it efficient. Once the code for the lessonware is written, the knowledge helps the tutorial to achieve the initial goals. We have seen that the student model is accurate and contains as much information as there are units (κ, σ) in the database. The accuracy of the marks allocated to each couple for a student depends on the number of problems covering that couple (κ, σ). If there are just a few problems, the students will soon know them by heart and will have the answers without really improving their skills. If there are many of them, it is possible to always bring new problems to the students. This implies that the database should contain as many problems as possible.

The nature of the problems has to be chosen carefully. The knowledge in the database must maintain a reasonable balance of problems between the categories, the concepts, and the levels. A good system presents the same number of problems by level and couple (Category, Concept) as discussed in Section 5.1.2.

5.1.2 Cubic Updating of the Database

The reader may have noticed that the knowledge in the database has a cubic organization and the three poles are: Category, Concept, Difficulty. We could improve the management of the database by creating a file updating the number of problems matching each triple (Category, Concept, Difficulty). Let us call this file a *Cube*. There are several advantages to this cubic organization:

- **Projection:** The database managers can project on any axis of the cube to obtain the total number of problem per level of difficulty, per category, per concept. Therefore there is a measure of the distribution of problems according to different criteria.
- **Maintenance:** The measures provided by the cubic organization allow the database managers to decide how to improve the knowledge for increasing the efficiency of the tutorial. An ideal organization should lead to the same number of problems for each triple (Category, Concept, Difficulty) of the problem. If the number of problems is not balanced, it may be customized for stronger players (problems in the difficulty range 0-2) or beginners (easier problems).

- **Cubic Updating:** Each time new problems are added to the database the Cube sho'ıld be updated accordingly. For a lesson (category, concepts, range of difficulty), IGoT has to pick up a few problems at random from the cube. When it does not know how many problems of the database match the parameters of the lesson, it has to scan the whole database and count them, then decide randomly which ones to include in the lesson. If the total number of available problems is known in advance, the random decisions can be made and it saves time.

The Cubic Organization has not been implemented in the first version of IGoT. We strongly recommend it for a new version especially after adding many problems in the database. Updating the Cube requires little time and saves precious seconds of repeated tasks when organizing the lesson.

5.2 In the Beginning

5.2.1 Learning the Rules

We have to wonder what we can do with this tutorial. Is it possible for a person that has never played Go before to learn just with the tutorial? The tutorial doesn't teach the rules of Go but we can argue that the rules are easy to learn without a computer. The advantage of the tutorial is that when learners master the basic rules, they can start using it to improve their skills.

Peter Witschital and Günther Stiege examine the novice's difficulties in ([Witsc 89], page 595). They explain the importance of a gaming environment for novices learning how to use a structured programming language. An intelligent and friendly coach is necessary to help novices who are confused when learning the basics of programming. In IGoT, the gaming environment is created by the timer and the scores in the student model.

The tutorial IGoT does not play Go but proposes problems and evaluates the player. We could wonder how someone evaluated 7 kyu by the system who has played with the tutorial compare to a Go player whose rank is estimated 7 kyu by the AGA[1]. The tutorial has to be seen as a tool to improve Go techniques but students should also play games to keep a balance and make use of their skills.

5.2.2 The Rules as a New Category

A beginner in Go could learn the rules with the tutor if we expand the limits of IGoT. One method is to create a new category of problems, called **Rules**, to explain the basic mechanisms. Some of the *concepts* to attach to this category are *capture in the corner, capture on the edge, in the middle, ko.*

5.3 Range of Levels

According to the previous section, any novice should be able to use IGoT. The professionals improve their skills in a different manner, with problems that require more reflection and discussion. They spend more time analyzing entire games. In other words, IGoT is a tool designed for amateurs.

[1]American Go Association

5.4 The Self-Correcting Method

The idea of a self-improving tutor has been discussed in the ITS literature [Kimbal 82], [O'Shea 79] and [O'Shea 82].

The expert has the difficult task of evaluating each problem. This task can be easier if one runs tests on students to evaluate the time and the difficulty of the problem. However, this is not an exact measure and the pedagogical knowledge attached to each problem is never exact. The goals of the tutorial should not be affected by this inexactitude, because the evaluation is closer to reality on the average.

We propose here a method to modify automatically the information concerning the time and the difficulty of the problem according to the performance of the candidates. We would allow the lessonware a write access to the database of problems to adjust data that differ from experience. An ideal system would allow for new problems to be added to the database. Then students would solve them and instead of updating just their associated model, the results would be reflected on the information contained in these problems. Precisely we can imagine a lessonware that updates the parameters of the linear relation between the strength and the time for each problem after a lesson. This technique would discharge the responsibility of the expert and correct the information he entered. In theory, the role of the expert can then be reduced to define the category and concept attached to a problem.

5.5 Application to Chess

Currently, we don't have any information from Gadwal [Gadwal 89]. Yet, the work done in IGoT can be generalized. The method applied to the game of *Go* can be applied to any domain where the nature of the knowledge is the same. The student modeling and its relation with the database of *Go* problems is probably a method that can be adapted to chess with the ELO rating system, that is the rating system used by the American Chess Association.

5.6 Publishing

IGoT can be used to publish customized lessons. The organization of the database prevents *Go* teachers from searching through books when in need of a lesson. They can now use IGoT in the manual mode to create a lesson in a specific category, with a particular subset of concepts and for a certain range of difficulty.

Conclusion

Many ITS projects never left the departments of research even after years of work by talented people. Therefore we must not expect big results. Some shortcomings and deficiencies are reasonable. But this first framework for $\mathcal{G}o$ must lead to a re-design of a more sophisticated version. This is the first step in the area of ITS for strategic board games. The $\mathcal{G}o$ library allows one to enter new problems. It is the first organized library of evaluated problems and could be used as a standard. The number of problems is still too small to reflect the potential of the ITS. When the database is bigger some players can start to use I$\mathcal{G}o$T for a year for instance, then come back to real games to see if the strength has increased. This is a natural and efficient way to test the system.

The books don't propose well organized problems, with a level, a category, and some concepts attached like in the database. The classification of the problems changes from one book to the other and the students don't always read books that are appropriate for their level. They often miss areas of $\mathcal{G}o$ knowledge where their deficiencies make the difference in close games.

By simply playing many games, players don't improve their level quickly, but there is a better improvement when the game is commented by an expert. Most players benefit from direct feedback about their moves.

The lessonware of the first version of IGoT is currently working although there are a few problems in the library. Even if allocating a level of difficulty to a problem is feasible, the question of evaluating the solution time needed is often more difficult. This encourages the implementation of a self-correcting system in the next version IGoT2.

The granularity of the student model and the choice of the units of knowledge opens a wide range of possibilities on the state of the student's ability (deficiency, strength and rating, weak category, concept). The easy management of this model and the quantity of information it contains seems to be a reasonable compromise.

The tests roughly confirmed our assumption for a linear relation time-rating on a *Go* problem. A test with many more *Go* players might have supported our assumption better. It would be worthwhile to investigate further the time/expertise relationship to see if the linear assumption is correct. This would certainly require much effort but help us to correct the problem of overrating or underrating that may occur in the current design.

We have investigated a new area of student modeling in tutorials for board games. There is still room for improvement but some lessons have been learned in designing and implementing IGoT and it can be used as a basis for further development of similar systems. Throughout the game of *Go* we experienced the difficulty of designing a knowledge communication system. IGoT belongs to the domain of *Exploratory software development*, as explained by Kierulf, Chen and Nievergelt in ([Kierulf 89], page 34). It proceeds by trial and error, along lines of *least resistance* or *greater* *promise* with a direction of achievement.

The main contribution of this thesis is to provide a good starting point for the development of ITS in a new domain where nothing has been done before, by introducing an original student modeling technique.

Bibliography

[Abayomi 89] Amos Abayomi David, Odile Thiery, Marion Crehange: Intelligent Hypermedia in Education, *ICCAL*, 1989, pp. 53–64.

[Brachma 85] Brachman, R.J.; and Levesque, H.J. (Eds) (1985) *Readings in Knowledge Representation*, Morgan Kaufmann, Los Altos, California.

[Brecht 88] B. Brecht and M. Jones. Student Models: The Genetic Graph Approach, *Laboratory for Advanced Research in Intelligent Educational Systems*, Department of Computational Science, University of Saskatchewan, Saskatoon, CANADA S7N 0W0. Aries@Sask, 1988.

[Burton 82] Richard R. Burton and John Seely Brown: An Investigation of Computer Coaching for Informal Learning Activities, in *Intelligent Tutorial Systems*, Sleeman and Brown (Eds), Academic Press, London, 1982, pp. 79–98.

[Conklin 87] Jeff Conklin: Hypertext: an Introduction and Survey, *IEEE Computer*, Vol.20, No.9, (September 1987).

[Elliott 89] Private communication of Chuck Elliott, Go teacher, president of the Go club of the University of Alberta, president of *Kakari Systems*, a consulting company specialized in databases, 1989.

[Gadwal 89] Private communication. Dinesh Gadwal, Graduate student, is currently developing an ITS for chess at *Laboratory for Advanced Research in Intelligent Educational Systems*, Department of Computational Science, University of Saskatchewan, Saskatoon, SK. CANADA S7N 0W0. Aries@Sask, 1989.

[GMag 88] Cho Chikun : *The Magic of Go, Ishi Press.*

[Greve 89] Steven H. Greve: A Real-Time Coaching Environment for Triangle Congruence Proofs, *ICCAL*, 1989, pp. 150-157.

[Hammon 89] Nick Hammond: Hypermedia and Learning: Who Guides Whom?, *ICCAL*, 1989, pp. 167–181.

[Holding 86] Dennis H. Holding: The Psychology of Chess Skill, *Lawrence Erlbaum Associates*, publishers, London, 1986.

[Kierulf 87a] Anders Kierulf: Proposed Standard File Format for Go Games, Dept. of computing Science, University of North Carolina, Chapel Hill, NC 27514.

[Kierulf 87b] Anders Kierulf: Computer Go and Artificial Intelligence, working paper, Dept. of computing Science, University of North Carolina, Chapel Hill, NC 27514.

[Kierulf 89] Anders Kierulf, Ken Chen, and Jurg Nievergelt: Smart Go Board and Go Explorer: A case study in software and knowledge engineering, in *New Directions in Game-Tree Search*, Workshop pre-prints, T.A. Marsland (ed.), Edmonton, Canada, 28th-31st May, 1989. pp. 33–52.

[Kimbal 82] Ralph Kimball: A Self-Improving Tutor for Symbolic Integration, in *Intelligent Tutorial Systems*, Sleeman and Brown (Eds), Academic Press, London, 1982, pp. 283–307.

[Loser 89] Marilyn Loser and Barry Kurtz: Quadratic Grapher: An Intelligent Tutoring System for Graphing Quadratic Equations, *ICCAL*, 1989, pp. 346–358.

[Megarry 89] Megarry, J. : Hypertext and compact discs - the challenge of multimedia learning, *British Journal of Educationnal Technology*, 1989, pp. 172-183.

[Nakhoon 89] Kim Nakhoon, Martha Evens, Joel A. Michael and Allen A. Rovick: CIRCSIM-TUTOR: An Intelligent Tutoring System for Circulatory Physiology, *ICCAL*, 1989, pp. 254–266.

[Nihon 73] The Nihon Kiin: The world's most fascinating game, vol.1 Introduction, *Sokosha Printing*, Tokyo.

[O'Neil 81] Harold F. O'Neil JR. Computer-Based Instruction: A State-of-the-Art Assessment, *Academic Press*, 1981.

[O'Shea 79] O'Shea, T. (1979a): Self-improving Teaching Systems: an application of Artificial Intelligence to Computer-Aided Instruction, *Birkhauser Verlag*, Basel, 1979.

[O'Shea 82] Tim O'Shea: A Self-Improving Quadratic Tutor, in *Intelligent Tutorial Systems*, Sleeman and Brown (Eds), Academic Press, London, 1982, pp. 309–336.

[Shira 89] Kiyoshi Shirayanagi: A New Approach to Programming *Go* - Knowledge Representation and Its Refinement, in *New Directions in Game-Tree Search*, Workshop pre-prints, T.A. Marsland (ed.), Edmonton, Canada, 28th-31st May, 1989, pp. 53–65.

[Sleeman 82] Sleeman, D.H.; and Brown, J.S. (Eds) (1982): Intelligent Tutorial Systems, *Academic Press*, London, 1982, pp. 83–86.

[Spada 89] Hans Spada, Michael Stumpf and Klaus Opwis: The Constructive Process of Knowledge Acquisition: The Student Modeling, *ICCAL*, 1989, pp. 486–499.

[Steel 60] Robert G. D. Steel : Principles and Procedures of Statistics. *McGraw-Hill*, New York.

[Uhr 69] Uhr, L. Teaching machine programs that generate problems as a function of interaction with the students. *Proceedings of the National ACM Conference*, New York, 1969, pp. 125-134.

[Wenger 87] Wenger E. Artificial Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge, *Morgan Kaufmann*, Los Altos, California, 1987.

[Witsc 89] Peter Witschital, Günther Stiege, Thomas Kühme: Experiencing Programming Language Constructs with TRAPS, *ICCAL*, 1989, pp. 591–602.

Related Works

[Ishi] ISHI PRESS INTERNATIONAL, 1400 NORTH SHORELINE BLVD., BUILDING A7, MOUNTAIN VIEW, CALIFORNIA 94043. TEL. 415-964-7294.

[GV1] Vol 1 Introductory Problems, *Ishi Press.*

[GV2] Vol 2 Elementary Problems (25-20 kyu), *Ishi Press.*

[GV3] Vol 3 Intermediate Problems (20-15 kyu), *Ishi Press.*

[G2] Basic techniques of Go *Ishi Press.*

[G6] Strategic Concepts of Go *Ishi Press.*

[G10] In the beginning, *Ishi Press.*

[G11] 38 Basic Joseki, *Ishi Press.*

[G12] Tesuji, *Ishi Press.*

[G13] Life and Death, *Ishi Press.*

[G14] Attack and defense, *Ishi Press.*

[G15] Endgame, *Ishi Press.*

[G18] What's your rating, *Ishi Press.*

[G24] Enclosure Josekis, *Ishi Press.*

[G29] Reducing Territorial Frameworks, *Ishi Press.*

[GW] Magazine Go World, published by the American GO Association.

[J-LD1] Life and death tesuji Vol1 (Japanese)

[J-LD2] Life and death Vol2 (Japanese)

[J-LD3]	Life and death Vol3 (Japanese)
[J-T1]	Tesuji Dictionnary Vol1 (Japanese)
[J-T2]	Tesuji Dictionnary Vol2 (Japanese)

Appendix A

The Game of GO

A.1 Introduction

In their paper [Kierulf 89], Kierulf and Nievergelt give an excellent introduction to the nature of the game reflecting both the simplicity of the rules and the complexity of the strategy.

GO is a two-person game. The players alternate placing a white and a black stone on some empty intersection of a 19 by 19 grid. Once played, a stone never moves but it can be captured and then removed as a prisoner. A player's objective is to secure more territory than his opponent, counted in terms of grid points. In the process of surrounding territory by laying down a border of stones that must be *alive*, fights erupt that typically lead to some stones being captured. Much of the difficulty of GO comes from the fact that during most of the game, few stones are definitely *dead* or *alive*. Their status can change during the course of the game. The game finishes when all the stones can be classified as dead or alive for sure. This philosophy of the game gives an idea of $\mathcal{GO}$, and we now explain the rules with more details.

A.2 Nine Basic Rules

The rules of $\mathcal{GO}$ are surprisingly simple. The fact that there are so few rules for a game as dynamic as $\mathcal{GO}$ is a proof that there is so much room for technical innovation and improvement ([Nihon 73], page 23).

Rule 1: A game of $\mathcal{GO}$ is played between two players.

Rule 2: One of the two players uses black stones and the other white. The players take turn in placing the stones on the board, one at a time. The first move (i.e., the first placing of a stone on the board) is made by Black (i.e., the black player). In a handicap game, however, White plays first.

Rule 3: The stone must be placed on one of the intersections.

Rule 4: A stone, once placed, cannot be moved (except when removed, Rule 6 below).

Rule 5: The player with more territory wins the game.

Rule 6: Stones whose *liberties* or "breathing spaces", have been completely filled are removed from the board.

Rule 7: No stone can be placed on an intersection where there are no liberties.

Rule 8: There are special restrictions on a player's moves in a *ko* situation.

Rule 9: Rules concerning handicap play.

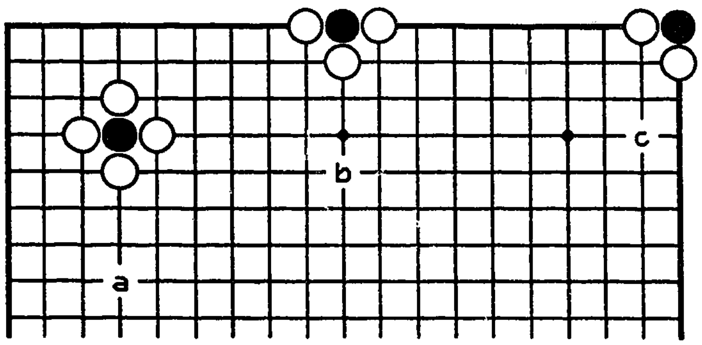


Figure A.20: The capture of prisoners.

Of these nine basic rules Rule 1 through Rule 4 should need no further explanation. The others are presented in the following section.

A.3 Captures

When all the adjacent intersections of a stone are held by the opponent's stones, the stone is captured and removed from the board. In situations *a,b* and *c* in Figure A.20 all the liberties of the Black stone have been preempted by White stones. At the same time that the last White stone is placed on the board, the Black stone is taken off the board by the White player. At the end of a game, the captured enemy stones are used to reduce the opponent's territory. This is done by replacing them on the board within the boundaries of the enemy territory before counting.

In a situation where, between two adjacent intersections, each side could alternatively repeat the move to capture the opponent's stone in an identical manner *ad infinitum* the rule is that the recapturing move cannot be done immediately. It is the situation of **KO** as illustrated by Figure A.21. If White plays *a* and captures

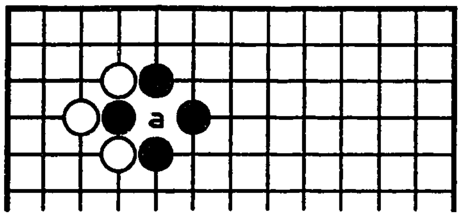


Figure A.21: The KO situation.

Black, then Black cannot recapture White immediately ([Nihon 73], page 28).

A.4 The Object of *Go*

In the game of *Go* territory is the final objective. The territory is counted at the end of the game and the player with more territories is the winner. The territory is made up of intersections which are so enclosed that no further invasion is possible. The territory is counted by the number of these intersections. Figure A.22 shows the end of a game on a 13 $\times$ 13 board. The unoccupied intersections make up the territory that belongs to either Black or White ([Nihon 73], page 24).

A.5 The Merits of *Go*

According to a certain well-known journal in Japan, the game of *Go* holds the first rank among all indoor amusements. The chief merits of *Go* as a game may be mentioned as follows ([Nihon 73], page 15):

1. It can be played at any time throughout the year.

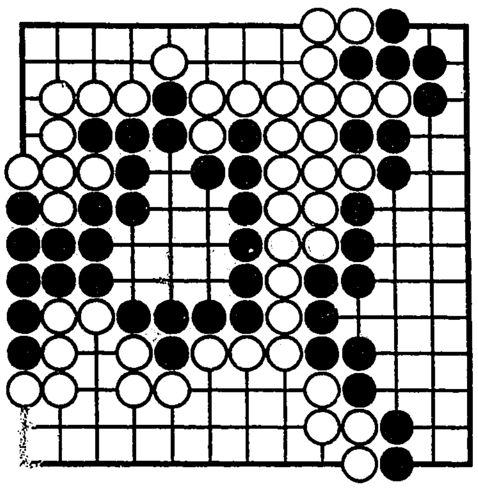


Figure A.22: The territories at the end of the game.

2. It does not need much room for playing.

3. It does not cause annoyance to other people.

4. The instruments, board and stones, are very simple.

5. It does not cost much.

6. The rules are easy to remember.

7. It can be enjoyed by people of all ages and both sexes.

8. A poor player can play with a good player by the handicap system.

9. It cultivates the power of thinking and is good for mental training.

10. It can be reproduced by keeping its records.

A.6 The Magic of GO

A presentation of the game would not be correct without an introduction to the numerous legends attached to it. We present now an explanation of the GO board ([GMag 88], page ix):

> *Generally speaking, when counting all things, one begins with the number one. There are on the GO board, three hundred and sixty intersections plus one. The number one is supreme and gives rise to the other numbers because it occupies the ultimate position and governs the four quarters. Three hundred and sixty represents the number of days in the [lunar] year. The division of the GO board into four quarters symbolizes the four seasons. The twenty-two points on the circumference represents the [five-day] weeks of the [Chinese lunar] calendar. The balance of yin and yang is the model for the equal division of the three hundred and sixty stones into black and white.*

Appendix B

The Rating System

B.1 Presentation

ap system allows a balance between players of different skill levels with-
the nature of the game [Kierulf 89]. The weaker player starts placing
2 to 13 stones on the board. The handicap system defines a fairly
SCALE. A weak player may be 20 kyu, implying that he receives
ones from a 10 kyu, who in turn receives 9 handicap stones from a
kyu. A first kyu takes black (plays first) against a first dan, who receives 5 stones from a 6 dan. This is as high as the amateur scale goes. Above that there is a separate scale for professional dan players, from the first to the 9-th. This scale has a finer grating. A difference of 3 levels corresponds to at most 1 handicap stone.

$$professional \left\{ \begin{array}{l} 9dan \\ \vdots \\ 1dan \end{array} \right.$$

$$amateur \begin{cases} 6dan \\ \vdots \\ 1dan \\ 1kyu \\ 2kyu \\ \vdots \\ 20kyu \end{cases}$$

To compensate for Black's advantage in even games, a **komi** of five points is added to white's score at the end of the game. Another half point is added often to award White the victory in case of a tie. In case of tie game with handicap white wins. Playing even games with komi, a player is expected to win one third of the games against a player one rank stronger, one ninth of the games against a player two ranks stronger. Thus two players quickly find their relative strength; by convention, the handicap is adjusted after winning three games in a row.

In the game of Chess, the American Chess Association maintains a rating with the **Elo-System** from Beginner to Master. Every chess player is on the same scale, while two scales are used for *Go*: The *amateur scale* and the *professional scale.*

B.2 The Algorithm of the Rating System

It is fairly well accepted that the probability of winning for a player A is an exponential function depending on the difference of handicap with player B [Elliott 89]. Every *Go* player has a score that reflects the strength. This score begins at −2000 for a beginner that is 20 kyu. Then, a score ranging from -1999 to -1900 corresponds to a 19 kyu player. A 5 kyu player has a score between -599 and -500. This score is updated after every properly recorded game[1].

When two players A and B have the same rank, the score of the winner increases by the same amount of points as the score of the loser decreases. We call Φ this value. Φ reflects the speed with which the players reach their rank. If it is too small then it will take many games to reach the rank above. If Φ is too big, then the evaluation of the ranks is not regular and players can see their rank changing every week, oscillating between two values.

When a player A plays against a player B one rank stronger and wins, we want to give him a reward better than Φ. If p represents the probability for player A to win in that case then, we want his score to be increased by $\Phi \times (1+p)$. If we assume that the function $\Delta R = f(\Theta)$ is of the form $\Delta R = \Phi \times e^{\alpha\Theta}$ then, the previous requirement leads to $\alpha = ln(1+p)$. The value of alpha reflects how big is the difference between two ranks.

In Figure B.23 we chose an arbitrary value $\Phi = 10$ and $p = \frac{1}{3}$. We represent the function $\Delta R = 10e^{0.288\Theta}$.

B.3 An Example

Instead of a long discourse, we present an example that should indicate the mechanism for recording the results of a game and updating the scores of both players.

[1]We mean that a game is properly recorded if it gives the number of handicaps and the result of the game.

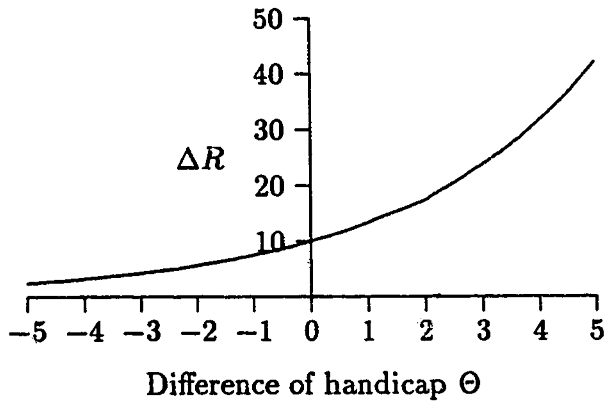


Figure B.23: The relation between *handicap* and *score*

Suppose Player A has a score of -1345 and player B -938. In other words player A is 13 kyu and player B is 9 kyu. Player A is weaker and therefore both players agree that player A gets 3 stones of handicap. The new handicap between the two players is now $\Theta = -938-(-1345)-(3\times 100) = 107$ in favor of player B that is still favorite. Suppose A wins, then his rating increases from -1345 to $-1345 + 10 \times e^{1.07\times 0.288}$ while player B's rating decreases by the same amount of points. If B wins, player A's score is decreased by $10 \times e^{-1.07\times 0.288}$ and player B's is increased by the same amount. On this simple example we notice that the reward is greater when a weaker player wins.

Appendix C

The Units of Knowledge in IGOT

Tables C.20 and C.21 included in this appendix summarize the different units of knowledge that are used in IGOT . For each concept we mention the corresponding categories that can be one of the six following ones: Endgame, Fuseki, Joseki, Life and Death, Shape, Tesuji.

Units of knowledge (Part 1)						
CONCEPT	E	F	J	L	S	T
Attach Tesuji - shape					1	1
Avoid thickness		1			1	
Bamboo connection (shape)					1	1
Bent four in the corner				1		1
Block					1	
Box six in the corner				1		1
Capture stones					1	1
Connect	1			1	1	1
Connecting stones - tesuji						1
Corner invasion		1				
Counting problem	1					1
Cut tesuji	1					1
Dead shape sacrifice of three stones				1		1
Dead shape sacrifice of four stones				1		1
Dead shape sacrifice of five stones				1		1
Dead shape sacrifice of six stones				1		1
Dead shape sacrifice of three stones				1		1
Descend to the 1-2 point				1		1
Descend towards edge tesuji						1
Double hane tesuji						1
Extend tesuji (nobi)		1			1	1
Extension limit		1			1	
Eye shape - destroy		1		1	1	1
Eye shape - how to make				1	1	1
Geta - surround tesuji						1
Hane at head of two stones						1
Hane at head of three stones						1
Hane tesuji						1
Knight's move - attack						1

Table C.20: List of the Units of knowledge (part 1).

Units of knowledge (Part 2)						
CONCEPT	E	F	J	L	S	T
Knight's move - shape		1			1	1
Knight's move - monkey jump	1					1
Knight's move - cut					1	1
Ko making tesuji		1		1	1	1
Kosumi - attack						1
Kosumi - connect					1	1
Ladder capture		1				1
Large knight's move tesuji		1				1
Live with seki	1			1		1
Miai	1	1				
Moyo construction		1				
Moyo reduction	1					
One space jump tesuji		1			1	1
Peep tesuji (nozoki)						1
Play beneath the stones				1		1
Push, cut, descend and throw in						1
Reduce territory	1					
Sacrifice living shape with false eye						1
Sacrifice two stones instead of one	1					1
Select best point	1	1				
Sente sacrifice tesuji						1
Snapback					1	1
Snapback to connect tesuji				1		
Spliting attack		1				1
Suicide rule tesuji				1		1
Throw-in tesuji	1			1		1
Wedge tesuji (warikomi)						1

Table C.21: List of the Units of knowledge (part 2).