

# The Dark Side of the Chessboard: an Assistant to Play Kriegspiel over the ICC

Paolo Ciancarini and Gian Piero Favini

Department of Computer Science, University of Bologna

**Abstract.** Kriegspiel is a variant of the game of Chess, introduced to make it more similar to wargames. It mainly differs from Chess in that players can only see their own pieces: they have to refer to a neutral agent – the umpire, or referee – to acquire additional, but always incomplete, information. This paper deals with the development of a visual interface to play Kriegspiel over the Internet Chess Club. The interface is intelligent insofar as it is able to interpret the referee’s messages obtained by the ICC server, offering the player an intuitive visualization of such messages, in terms of graphic abstractions which represent the invisible pieces of the opponent. In this sense the interface is able to let light into the dark side of the chessboard.

## 1 Introduction

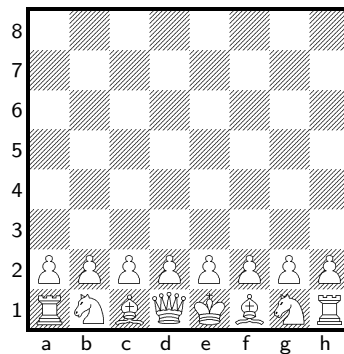
When the first chess playing programs were designed, their interfaces were textual. The advent of graphical user interfaces allowed to design more natural interfaces. In fact, all modern chess playing programs offer a 2-D or 3-D interface simulating a chessboard.

In this paper we study the development of a graphical user interface for Kriegspiel. Kriegspiel differs from Chess by hiding from each player his opponent’s moves: in fact the two players play on different boards. A player only knows the position of its own pieces, while his opponent’s pieces are “in the dark”, ie. they are invisible. A referee evaluates a player’s try and replies to both players with a message that contains some information, but not the real move.

Kriegspiel was invented more than one century ago, and it is currently quite popular on the Internet Chess Club (ICC). We intend to design a special interface for playing Kriegspiel over the ICC. Why this is an interesting problem? Kriegspiel, also called sometimes *Invisible Chess*, has several features which require a sophisticated interface.

Kriegspiel is very similar to Chess. In fact, most Chess rules are valid in Kriegspiel [7]. Thus an interface to play Kriegspiel can be trivially obtained adapting an interface for Chess. In fact, on the ICC the interface to play Kriegspiel is similar to the one for playing Chess: the only difference is that the opponent pieces are not shown.

Thus, what a Kriegspiel player typically sees when he starts to play a game as White is the diagram shown in Fig.1, where Black pieces are invisible.



**Fig. 1.** A trivial interface for Kriegspiel: the opponent pieces are invisible.

Then there is the issue of the referee. The first programs for Kriegspiel focussed on designing a referee for playing over a local area network [2, 9, 10]. The current implementation of the referee in the Internet Chess Club is in some sense a descendant of those programs. Thus, a Kriegspiel interface usually requires an additional window to display the announcements from the referee (which are always broadcast to both players).

Each player has to keep in mind all the information received from the referee. When playing Over The Board (OTB) Kriegspiel, the players can use a complete set of pieces, which are used to build hypotetic positions compatible with the referee's announcements. In fact, in actual OTB Kriegspiel games the players are allowed to place the opponent pieces as they prefer, trying to infer useful information from the referee's messages.

Thus, we try to design a more effective Kriegspiel interface: it should be able to interpret the referee's announcements offering a graphical interpretation in order to help its user. Our aim in this design effort is to fully specify the behavior of a Kriegspiel agent able to interpret the referee's messages in terms of constraints on chess positions. Such an agent will be a module of a more general program able to play a complete game of Kriegspiel.

This paper has the following structure: In Sect. 2 we describe the rules of Kriegspiel; in Sect. 3 we offer a graphical interpretation of the referee's messages; in Sect. 4 we define the specification of the user interface; In Sect. 5 we describe the low-level interface to the ICC server; finally, in Sect. 6 we draw our conclusions.

## 2 Kriegspiel

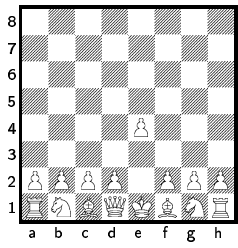
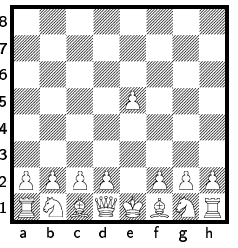
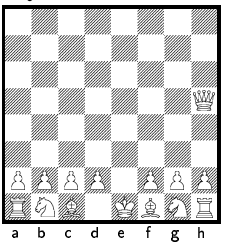
We will describe shortly the rules enacted on the Internet Chess Club [5]. Every rule in orthodox Chess also applies to Kriegspiel, with two exceptions: the game will *not* result in a draw because of a position being repeated three times, nor will

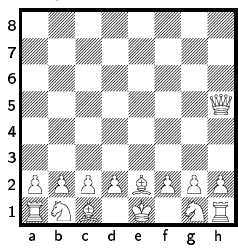
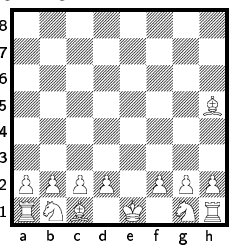
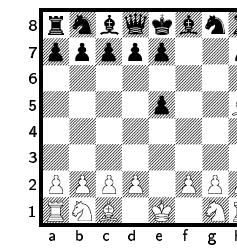
a draw occur due to the lack of captures or pawn actions for 50 moves in a row. Aside from these exceptions, the major innovation of Kriegspiel over orthodox Chess is that players cannot see their opponent's pieces, and they have to make guesses concerning their location.

Given the incomplete-information nature of Kriegspiel, a new agent is introduced to act as a broker between the players: the umpire, or referee. Only the umpire has access to complete information, and it performs all communication with players (players cannot directly interact with each other in any way). Before each player's turn, the umpire transmits a message containing additional information (note that, although the message concerns the current player, both players actually receive the message). This information includes:

- Number of pawn tries for the player on move. This is defined as the "number of legal capturing moves using pawns". Pawn tries include possible en passant captures.
- A capture happened as a result of the last move. The message specifies where the capture took place and whether the captured piece was a pawn or another piece (in this case, the actual piece is not revealed).
- The current player's King is in check. The message also reveals the check type, which can be "rank", "file", "short/long diagonal" (from the King's point of view), or "knight".

A whole game could be displayed to the White player as follows:

1.e4	2.e5	3.Qh5
		
	One pawn try, Pawn captured in e5	Check on short diagonal

4.Be2	5.Bh5	
		
One pawn try, Piece captured in h5	Pawn captured in h5 Check on short diagonal	Checkmate

The last diagram shows the final position which appears when the game terminates.

In addition, if a player attempts an illegal move, the umpire refuses such a move and will notify that player that he should make another move instead. Thus, illegal moves are a means to acquire information on the state of the chessboard, and it is possible to deliberately try a move which is likely illegal, with the purpose of gaining such an information. On the ICC the opponent will receive no notification of a player's illegal moves, and this increases the asymmetry between the players, meaning that either player knows what he knows, but he does not know what his opponent knows.

### 3 The interface

We have developed a Java visual interface for playing Kriegspiel over the ICC. It consists of an extensible three-layer framework, *JVariant* [8], aimed at providing specific support for as many variants of chess as possible. *JVariant* was developed with Kriegspiel as its primary focus, but its flexibility makes it easy to develop visual support for completely different chess variants by simply implementing a single Java interface. The three layers allow to decompose the tasks into logical sublevels, thus separating the visual aspects from other, behind the scenes issues, such as rules management and communication protocols.

As shown in Fig. 2, the layers are as follows:

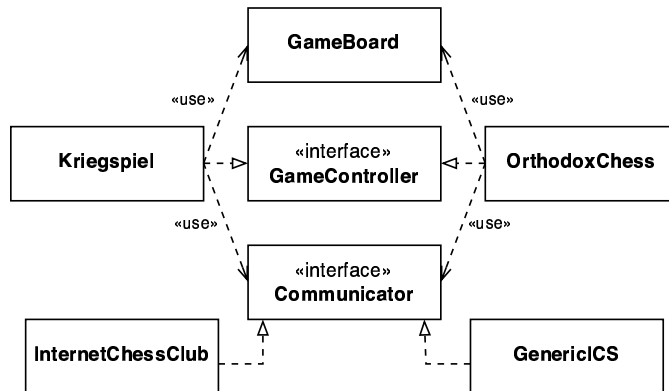


Fig. 2. UML class diagram representing the three-layer architecture of *JVariant*

- **Communicator**. This interface rules information exchange with the remote communication endpoint. For example, an implementation of this interface

provides protocol support for the ICC server. The higher levels remain oblivious to the inner workings of the implementing class; they only use abstractions such as moves and other game-specific information (sets of tag-value pairs listing additional information that does not fit in a move, like pawn tries in Kriegspiel).

- **GameController**. This interface provides the contract for dealing with a chess variant. Implementing classes will have to specify rule enforcement mechanisms and user feedback behaviors that are specific to the variant being played. Computer-assisted Kriegspiel is one of the most complex variants, and its GameController contains, among other things, the algorithms for the visual feedback described in the remainder of this paper.
- **GameBoard**. The actual object manipulated by the end user. A GameBoard is a passive object provided with the basic behaviors of a normal chessboard; it will delegate all decisions to the underlying GameController, which can also heavily customize the board's graphical skeleton to adapt it to the variant being played. For example, a minimum requirement for a Kriegspiel interface is to have a counter to keep track of captured enemy pieces. This is created by the GameController and attached to the GameBoard.

## 4 Visual feedback for Kriegspiel

Kriegspiel makes for a tough challenge as far as visual interfaces are concerned. An intelligent interface should provide as much information as possible, and in an intuitive, unobtrusive, constructive manner. There is no point in filling a window with data if the player finds it distracting enough to cancel its benefits. The aim of a Kriegspiel interface should be that of keeping the player focused on the game, and relieving him of as much memory effort as possible.

At the same time, we intend the specification of our interface as a study in the semantics of the referee's announcements. In another paper [4] we have defined and discussed the notion of Information Set, that is the set of pieces of information that a player can logically infer from the referee's messages. Our main requirement is that all messages from the referee must have a graphical interpretation on the interface, consistent at least with the position of visible pieces. The consistency should be enforced also with respect to the past messages got from the referee, but this is more difficult, as we will see. Interestingly, Kriegspiel is used in a recent book on object oriented design [11] as a case study, but no analysis of the interface is given. In any case, we intend our study as an exercise in knowledge engineering similar to the one described in [6].

Last but not least, we require that the interface is able to play other Kriegspiel-like variants, like Invisible Chess [1], where only single pieces can become invisible, or ShadowChess [3], where the enemy pieces which come under attack are made visible.

To address these requirements, we developed a module that extends the traditional iconography of the game of Chess by introducing three new pseudo-pieces, called *guess tokens*, as shown in Table 1.

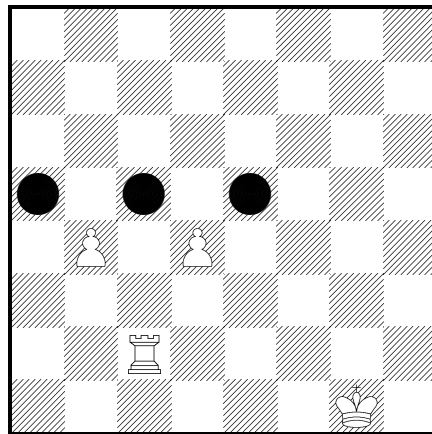
	Piece	Check	King
White	○	⊕	♔
Black	●	⊕	♚

**Table 1.** Kriegspiel guess tokens, as implemented in JVariant.

Guess tokens are all circular in shape. Also, some guess tokens, specifically Piece tokens, may appear at various degrees of transparency on the chessboard. The more opaque the token, the higher chance there is that an enemy piece actually be in that square.

- The **Piece token**, carrying no other distinguishing sign, simply informs that a generic enemy piece might be hiding there. It is activated in response to pawn tries and captures: in the former case, all the possible target cases are highlighted; in the latter, a fully opaque token is put on the case where the capture takes place, meaning that the presence of an enemy piece there is now certain.

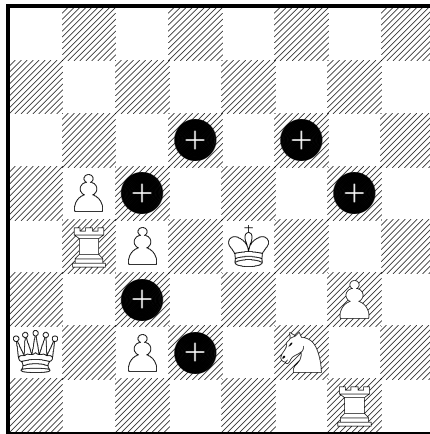
Let us demonstrate the usage of Piece tokens in regards to pawn tries. **Figure 3** depicts what JVariant would show to the White player, if the referee says "Two pawn tries". This diagram, as well as the following ones, portrays what JVariant displays, knowing the position of its pieces as well as the most recent umpire message.



**Fig. 3.** The umpire says: "Two pawn tries".

- The **Check token** warns a player that his King is in check, and the offending piece might be found on the selected case. For example, in **Figure 4**, the White King finds itself threatened by a Knight. Later we will show that it is sometimes possible to narrow down the choices and highlight a lower number of cases.

In order to facilitate players in using the interface, the Check token is identified by a '+' sign, which is universally known as the symbol for check positions in all literature and game transcriptions.



**Fig. 4.** The umpire says: "Knight Check".

- The **King token** is the counterpart to the Check token; when a player puts the enemy King in check, these tokens list the locations where it might be hiding. As it is described later, placing this kind of token raises the most problems. **Figure 5** depicts a possible disposition for the Black pieces, causing the check in **Figure 4**.

Now that the basic icons for the interface have been introduced, remains to show how they can be used to provide non-trivial, immediate information to the user. Our interface supports the following results with a few small exceptions that will be explicitly pointed out, and the implementation of these properties took the better part of the time resources devoted to the project. In fact, the complexity of the situations originating from the seemingly laconic messages of the umpire was somewhat underestimated at the beginning of the development cycle; while the umpire can only generate a handful of messages (eight including the five check types, plus the notification of illegal moves), they can assume different hidden meanings and their combinations make for additional interesting cases. This partly explains why strong artificial players for this discipline have not been programmed yet.

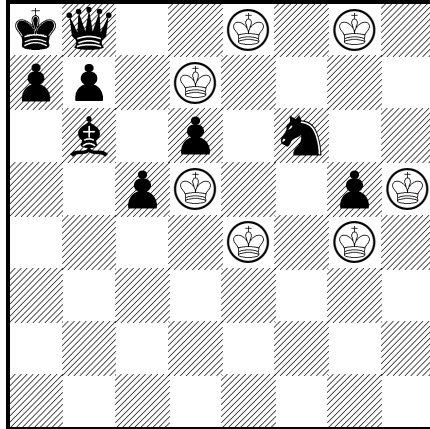


Fig. 5. Black plays ... Nh7-f6, the umpire says: "Knight Check".

#### 4.1 The Piece Token

As stated above, this token makes its appearance whenever pawn tries or enemy captures take place. Pawn tries are vital tools in finding out the disposition of opponent pieces, which makes pawns extremely precious in Kriegspiel: they are the only piece able to gather information on their surroundings without attempting to move.

The trick of putting Piece tokens wherever pawn captures might happen is a trivial, but effective one. It provides immediate information, and the player can often guess, if not the exact capturing move, at least the most likely area on the chessboard. *This method is partially inaccurate in the case of en-passant captures, since the location of the captured piece does not actually coincide with that of the token, but for the purposes of Kriegspiel, this is rather unimportant.*

It would be possible to highlight **opponent pawn tries** as well. Each time the opponent receives a pawn try notification, the program would then place a token on every legal case which might host the pawn. However, this would probably clobber the chessboard with pawn tokens, so it was decided not to implement this function. JVariant will still warn a player textually whenever his or her opponent gets pawn tries.

The other instance of Piece token insertion happens upon opponent capture. A token with a 100% likelihood (fully opaque) replaces the captured piece. This token will then fade out with each subsequent move, representing the odds of the piece still being on the same case reducing over time. Since the opponent is aware that the location of the offending piece is now well-known, he or she will tend to move it again if possible, in order to return it to the shadows. Because of this tendency, this type of token fades out quickly, its opacity being halved on every opponent move. When a fixed threshold is crossed, or a friendly piece "explores" the case, the token is removed from play.



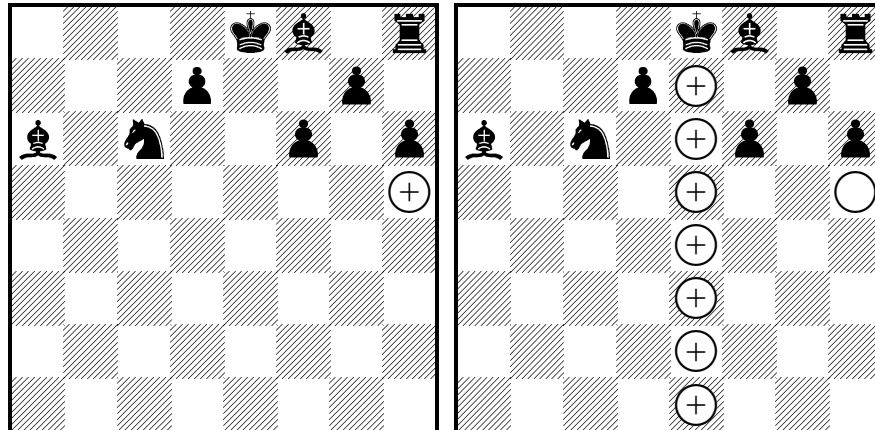
Pawn tries, or lack thereof, can also interact with the other two tokens to help narrow down choices. These techniques are described in the following sections.

## 4.2 The Check Token

The Check token highlights the possible locations of the piece attacking the player's King. Its basic behavior is quite trivial: the program simply stamps tokens starting from the cases adjacent to the King until a friendly piece or an edge of the chessboard are met. As it is easily seen, Knight checks work in a slightly different way, and only compatible empty cases get touched in this case.

However, if one remembers the rules of chess, there is much more information which can be extracted from a check notice.

- **If a capture took place on the last move...** Here, there are two basic cases. The former happens when the capturing piece lands into a case and directly threatens the player's King. The latter happens when the capturing piece uncovers another piece behind itself, clearing its 'line of sight' towards the player's King. These two cases seem to be easily recognizable from one another; it suffices to **determine whether the case in which the capture took place is compatible with the check type being revealed by the umpire**. In other words, if the player is told his or her King is under File check and the capture did not happen in the same file as that of the King, then that piece has nothing to do with the check. On the other hand, if the capture actually took place in the right File, then the offending piece must be responsible for the check, and only the target square must be lit with the Check token, as exemplified in **Figure 6**. It is to be noted that, while our Kriegspiel interface will display what is shown in the diagrams, this is by no means the most detailed information we can achieve. In fact, it will be readily seen that in the diagram to the right, the offending piece cannot be anywhere but in e1, and the piece that performed the capture in h5 must be a Bishop. The reasoning is as follows: the only legal moves originating from column 'e' and ending in h5 are ♖♔e5-h5 or ♘♔e2-h5. But neither a Rook nor a Queen could have been there, or we would have been in check already! Therefore, only ♘e2-h5 is plausible.
- **If pawn tries are possible, then the attacking piece, and only it, can be captured.** As stated in the rules of chess, there are three ways a player can react to a check: by moving the King out of enemy range, by moving a piece to cover the King, or by capturing the offending piece. This means that, if pawn tries can be performed, the attacking piece can be the only target. This narrows down the possible locations to those that are compatible with the current check type *and* a possible target of a pawn try. Most of the time, only one or two cases will meet these requirements, and the location of the enemy piece can be accurately described.  
An example of this technique is given in **Figure 7**.
- As an additional rule, **if pawn tries are not possible, the attacking piece cannot be in a capturing position for any pawn, unless that**



**Fig. 6.** In both diagrams, the umpire announces a capture in h5. In the left diagram, the umpire says "short diagonal check". In the right diagram, the umpire says "file check".

**pawn is protecting the King from another piece.** The wording of the previous sentence is actually more complicated than the rule itself: it simply means that, if no pawn tries are announced by the referee, and the player's King is in check, in general the attacking piece cannot be attacked by the player's pawns *unless* that pawn is being blocked by another piece. Because a picture is worth a thousand words, **Figure 8** explains this fact intuitively, showing that, in Kriegspiel, nearly every rule has its exceptions. The red pawn might be blocked by the possible presence of a Bishop or a Queen on the King's diagonal. On the other hand, the blue pawn has no constraints binding it to its King, and as such the lack of pawn tries attests that the piece giving check cannot be found in c5.

This particular case has not been implemented in the current release of JVariant, as its benefits did not justify the time resources needed for its coding. It is being explained here to convey the subtlety and complexity of the information that one could elaborate basing themselves on umpire messages.

### 4.3 The King Token

The King token is complementary to the Check token; while the latter shows up whenever the player's King is in check, the former makes its appearance if the **opponent's** King is in check. As mentioned earlier, the King token offered a harder challenge to the interface developer, because the check notifications sent by the umpire are King-centered, that is, they describe the type of check basing on the location of the attacked King, which is, of course, generally unknown

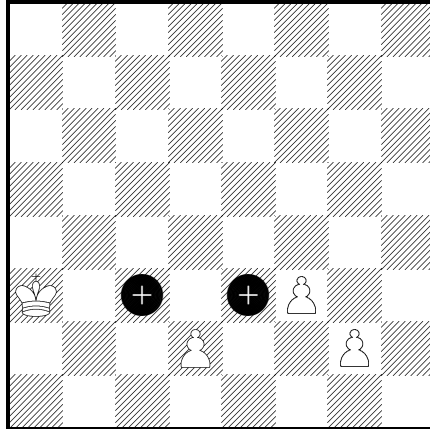
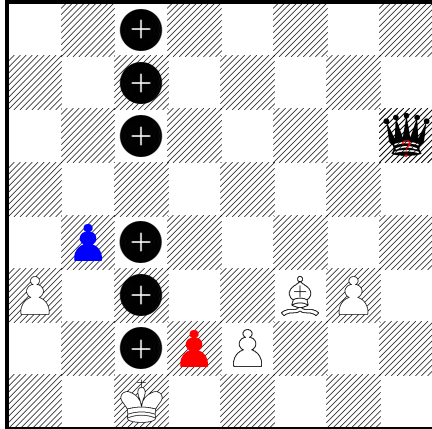


Fig. 7. The umpire says "Rank check, 1 pawn try"

to the opponent and their playing interface as well. This raises a number of problems, most of which occur when **diagonals** are involved.

- The first problem consists of **determining what allied piece is attacking the opponent's King**. Note that this problem is in many ways specular to the one pointed out in the last section, when dealing with captures (**Figure 6**). Only, this time the dilemma does not limit itself to the 'capture+check' combination; it is a constant presence that the interface has to deal with.
- The basic approach to this issue is analogous to the one adopted in the last section; that is, the program **checks whether the last moved piece is compatible with the umpire's information**. In other words, if the player moves a Rook and the umpire notifies a long diagonal check, then there is no way a Rook could have caused such a check. There must be another allied piece capable of a long diagonal check, previously hidden by the Rook. Unfortunately, what if the Rook was hiding **two** Bishops on two different diagonals? **In general, it is not always possible to determine what piece is causing a check**. See **Figure 9** for a visual example. In cases such as this, the interface will have to select all applicable pieces and consider all the possible choices.
- While the previous statement proves that the placement of King tokens is significantly more challenging than that of Check tokens, it is often possible to rule out several location through relatively simple algorithms. First and foremost, any **diagonal check** can be either short or long, and if a location belongs to the wrong diagonal, it has to be discarded. Additionally, if the attacking piece is the one which has just been moved (or so believes the program), **the diagonal along which it has moved is to be ruled out**, since the diagonal was already under friendly control. It should be noted that **this does not apply if a capture took place after the last move**,



**Fig. 8.** The umpire says "File Check".

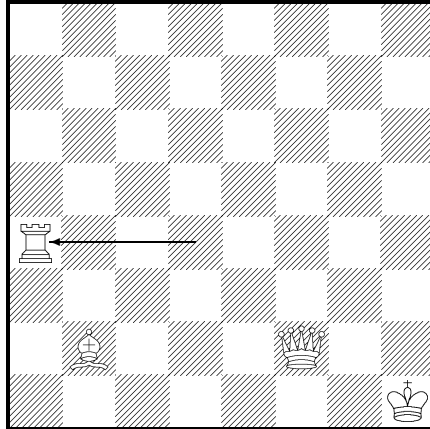
since the King might be found beyond the captured piece. In this case, only the half diagonal between the original position and the new position of the attacking piece is to be excluded, as shown in **Figure 10**.

- A very peculiar case occurs whenever the umpire announces a **double check**. This happens if a King is threatened by two pieces at the same time, and leads to an interesting, and possibly unexpected, result. Under these circumstances, **it is often possible to pin-point the exact position of the enemy King**. The reason is quite simple: the two sets of King tokens generated by the two pieces (one of which might actually be the union of two different pieces, because of the undecidability shown in **Figure 9**) have to be **intersected**, which will usually narrow the choice down to one or two cases at most. For instance, in **Figure 11**, the umpire announces two ongoing checks at the same time, thus mercilessly revealing the location of the opponent's King.

## 5 The Communication Layer

The Communicator layer is the lowest level in the JKrieg hierarchy. As the name implies, this interface is able to communicate with another entity, called an **endpoint**. Depending on the actual endpoint, the communication protocol and techniques will vary. We have used socket data transmission to play over the Internet Chess Club, but virtually anything is possible so long as the Communicator interface is respected, that is, if all the required methods are implemented.

One of the main aims of the Communicator layer is to consume incoming move notifications, in turn producing standardized diagrams which can be consumed by the higher layers, hiding the nature of the remote endpoint to the overlying GameController. This is accomplished through the **VirtualBoard** class. A



**Fig. 9.** Following Rd4-a4 ...the umpire says "Long diagonal check". Which one is guilty, the Bishop or the Queen?

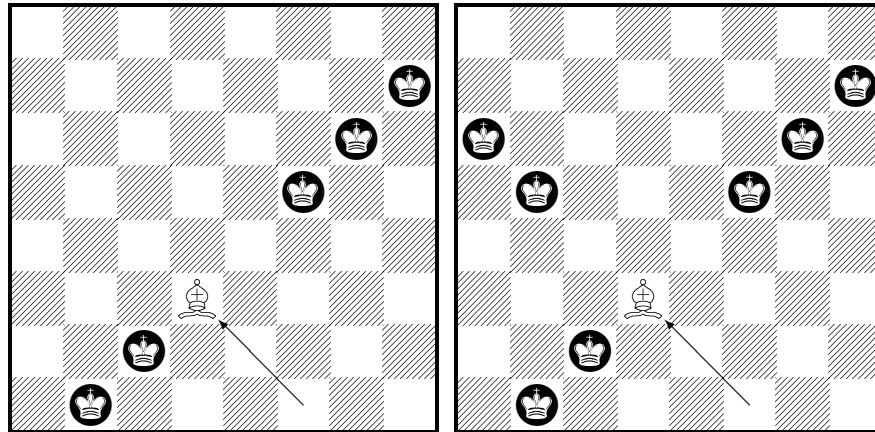
VirtualBoard object provides an internal, uniform representation of the pieces on a chessboard at any given time. It also provides a few additional useful features, like telling whether a given case is being controlled by a player.

VirtualBoard objects are the medium through which incoming moves are shipped to the GameController. Instead of just notifying about the move, the Communicator sends the updated chessboard. The goal here is to ensure maximum scalability and expandability: there are a few chess variants where the concept itself of "move" is a very peculiar one. Kriegspiel is one example of it, as the player technically receives no moves at all, but there are more situations where this issue shows up. For instance, in **Progressive Chess**, the first player performs a move, then the opponent makes two, then three, and so forth. So long as a few rules are followed (for example, only the last move in a sequence may give check), any array of moves can be played. However, VirtualBoard objects alone may not be sufficient to describe the situation on the gaming board. There might be additional information that cannot be directly stored within such objects, such as the remaining time, or the umpire messages in the case of Kriegspiel.

### 5.1 The ICCDriver class

The **ICCDriver** class is the Communicator implementation allowing play over the Internet Chess Club. After a brief overview of the ICC's history and features, further details shall be given concerning the inner working of this class.

The ICC is the oldest of Internet Chess Servers, dating back to the eighties, when a small community of chess players foresaw the dramatic impact that the Net could have on the game of chess. Today, it is by far the largest chess community to be found online, often simultaneously hosting over two thousand players.



**Fig. 10.** In both diagrams, White plays Bf1-d3 and the umpire says "Long diagonal check". In the diagram to the right, the umpire also announces a capture in d3.

Its code was completely rewritten in 1992, still maintaining full backwards compatibility, to improve in efficiency and compensate for the ever growing number of players.

Basically, the ICC is a server that allows players to log on and find opponents to challenge. It supports player rating using the **ELO rating system**, which lets a player keep track of his or her performance in time. Aside from orthodox chess, over twenty variants can be played, and it is also possible to watch and review other people's games. On top of that, it offers the full capabilities of an IRC-like chat system. Most of these features are only available to registered members who pay a monthly fee, though login as a guest is allowed, thus letting anyone play an unrated game.

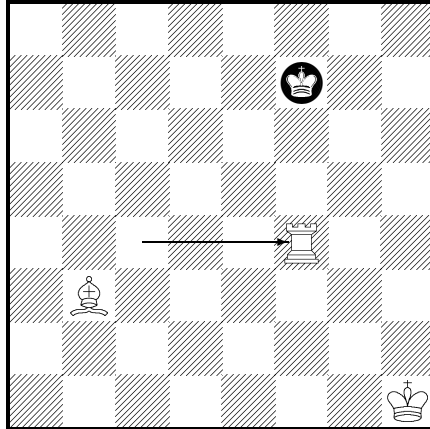
The Internet Chess Club makes use of a human-readable protocol. In fact, it is perfectly possible to log on and play games via a telnet client without the need for a graphical interface, even though the user experience is bound to be lacking and incomplete. Of course, the programmers introduced a mechanism to make data transmission a more program-friendly process; this is accomplished through the concept of **datagram**. In order to have the server send datagrams instead of the usual stream of messages (which is much harder for a program to parse), the following command has to be issued.

```
set level 11
```

After the server has been instructed to do so, it will start to format its messages in a different fashion. The structure of a datagram is as follows.

```
[ command-number player message ]
```

<sup>1</sup> Level 2 datagrams are available on ICC, but their purposes and usage go beyond the scope of this paper.



**Fig. 11.** White plays Rc4-f4 . . . , the umpire says "File check, Long diagonal check".

Here, `command-number` is an opcode representing the incoming message type, `player` indicates the username of the player the message come from, and `message` contains the actual message text. This way, the program can figure when a message ends, what it is meant to do, and who sent it in the first place. There is a number of available opcodes, though JKrieg only handles a handful of these, simply trashing and ignoring the rest, as they are of little relevance to the game itself. **Table 5.1** shows a selection of command opcodes implemented in JKrieg, together with their meaning.

Opcode	Meaning
2	<b>Move.</b> Represents a chessboard update during a game.
19	<b>Disconnection.</b> Opponent has disconnected from the game.
101	<b>Tell.</b> Privately sends a message to a single user.
108	<b>Forfeit.</b> User forfeits due to lack of time.
110	<b>Say.</b> Sends a message to everyone in channel.
121	<b>Issuing.</b> Match request sent to selected opponent.
123	<b>Accepted.</b> User has accepted match request.
207	<b>Draw.</b> Game has ended in a draw.
208	<b>Resign.</b> User resigns and concedes the game.

**Table 2.** Some ICC opcodes, as implemented in JKrieg.

The message body contains all the relevant information. When the datagram transmits a move (more precisely, a **ply** or **half-move**), the body is an ASCII representation of a chessboard, with letters such as K and P standing for the various pieces. Uppercase letters represent White pieces, whereas lowercase let-

ters are Black pieces. The datagram also contains time information, showing the clock for both players.

There is a method, `produceBoardFromText`, that transforms the ASCII-based chessboard into an equivalent `VirtualBoard`, ready to be consumed by the higher levels. This method, as well as other methods in the `ICCDriver` class, makes use of a support class, `RegexpChecker`, which implements a very simple parser for a subset of regular expressions. The `RegexpChecker` never needs to be instantiated, because all of its methods are declared as **static**.

The `ICCDriver` receives datagrams from the Internet Chess Club via the `Socket` class inside the `doIOLoop` method, required by the interface contract. The socket returns bytes as they become available from the network connection, sleeping while the connection is idle (a useful feature of Java, preventing the program from polling the socket and wasting precious processor cycles). These bytes are then stored in a temporary buffer until a complete datagram can be extracted from it. The datagram is delivered to the `parseMessage` method, which will take appropriate action and usually invoke the driver's `GameController`.

In addition to `Communicator`, `ICCDriver` also implements the `Runnable` interface. This basically means that `ICCDriver` can be **run** as a separate thread, and it will just execute the `doIOLoop` method (which is basically the typical infinite input/output loop) until the thread itself is killed by the user quitting the application.

## 6 Conclusions and Future Works

Our work is motivated as a study on the importance of an intuitive visual system as a way of delivering information that can then be used as a factor in higher-level decision making. We have showed that even a small set of possible messages can lead to unexpected complexity that is best left to the computer to handle, so that the human player can focus his attention on the overall strategy. This being said, we have done little more than just scratching the surface of what `JVariant` can potentially do. At this time, for instance, the program is only able to acquire a subset of the information a human mind can infer from a given set of positions and umpire messages. The `Kriegspiel` module is currently limited in two important areas: it can derive very little information from illegal moves, and it has no memory of the past, that is, it bases its assumptions on the latest position alone.

Illegal moves are what makes `Kriegspiel` intriguing as a metaphor of real world incomplete information situations involving conflict and competition. A legal move, when it does not trigger a message (that is, the umpire stays 'silent'), adds very little information to the player's knowledge base, aside from the obvious fact that there were no enemy pieces in between (and not even this in the case of a Knight). An illegal move, on the other hand, can potentially probe the chessboard not unlike a radar, to the point that attempting moves that are almost certainly illegal in order to discover more of the chessboard may become a viable strategy, though it involves some risk.



## References

1. A. Bud, D. Albrecht, A. Nicholson, and I. Zukerman. Playing “Invisible Chess” with Information-Theoretic Advisors. In *Proc. 2001 AAAI Spring Symposium on Game Theoretic and Decision Theoretic Agents*, pages 6–15, California, USA, 2001. American Association for Artificial Intelligence.
2. J. Burger. UMPIRE: An automatic kriegsspiel referee for a time-shared computer. In *Proc. 22nd ACM National Conference*, pages 187–193, Washington, USA, 1967. Association for Computing Machinery.
3. Dark Chess. <http://www.itsyourturn.com>.
4. P. Ciancarini, F. Dalla Libera, and F. Maran. Decision Making under Uncertainty: A Rational Approach to Kriegspiel. In J. van den Herik and J. Uiterwijk, editors, *Advances in Computer Chess 8*, pages 277–298. University of Limburg, Maastricht, The Netherlands, 1997.
5. The Internet Chess Club. <http://www.chessclub.com>.
6. A. Kierulf, K. Chen, and J. Nievergelt. Smart Game Board and Go Explorer. A Study in Software and Knowledge Engineering. *Communications of the ACM*, 33(2):152–166, 1990.
7. David Pritchard. *The Encyclopedia of Chess Variants*. Games & Puzzles Publications, 1994.
8. The Sourceforge JVariant project page. <http://sourceforge.net/projects/jvariant/>.
9. C.S. Wetherell, T. Buckholtz, and K.S. Booth. A Director for Kriegspiel, A Variant of Chess. *The Computer Journal*, 15(1):66–70, 1972.
10. C.S. Wetherell, T. Buckholtz, and K.S. Booth. A Program to Referee Kriegspiel and Chess. *The Computer Journal*, 18, 1975.
11. R. Wirfs-Brock and A. McKean. *Object Design. Roles, Responsibilities, and Collaborations*. Addison Wesley, 2003.