Automatic Acquisition of Tactical Go Rules

Tristan Cazenave
LAFORIA-IBP
Université Pierre et Marie Curie
4, place Jussieu
75252 PARIS CEDEX 05, FRANCE
cazenave@laforia.ibp.fr

Abstract

Gogol is a rule-based computer Go program. It uses a lot of reliable tactical rules. Tactical rules are rules about simple goals such as connecting and making an eye. Gogol uses a simplified game theory to represent the degree of achievement of the goals. The automatic acquisition of tactical rules follows a three step process : Pattern generation, Game evaluation, Generalisation. Gogol has metainformation about the correct use of the rules in global Go positions.

Key words : Game of Go, Machine Learning, Metaknowledge,Knowledge Acquisition, Game Theory.

1 Introduction

Gogol is a computer Go program which uses symbolic rule-based techniques. It uses thousands of rules to know if some tactical goals can be achieved. This paper describes these tactical rules and how they have been automatically acquired. It should interest every Go programmer using a rule-based program. It provides them with a complete and reliable set of tactical rules.

Most of the algorithm developed to learn in Go are Similarity Based Learning algorithms [Stoutamire 1991], [Pell 1991], [Enderton 1991], [Schraudolph 1994], [Brügmann 1993], [Enzenberger 1995]. I believe Go is a domain which should better be approached by deductive learning algorithms such as those described in [Kojima 1994] and as the approach described in this paper.

The first part of the paper is concerned with the representation of rules using a simplified game theory. The second part is about the learning process : automatic generation of patterns, evaluation of these patterns, generalisation of the resulting rules. The third part shows how these rules should be used in a Go program. The extension of the work is discussed and perspectives for future work are shown. A short representative listing of the rules is given in the annex.

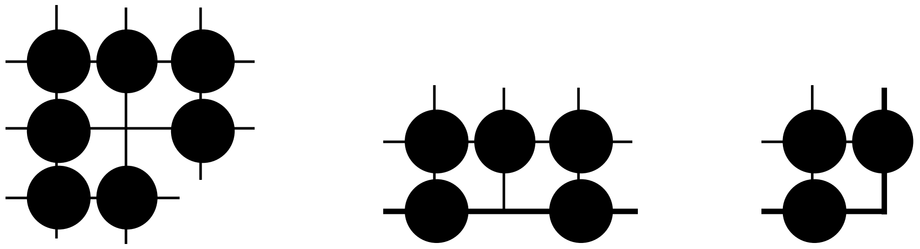
2 The rules

2.1 The goals concerned by the rules

The rules concern two goals : **Make an eye** and **Connect** two stones.

In all the paper, Black is assumed to be the friend colour and white the enemy colour.

The goal **Make an eye** is reached when one of these three patterns is recognised :



Using rotations and symmetries, there are 12 patterns to match.

The goal **Connect** is reached when two stones are part of the same block. Gogol uses a recursive function to decide whether two stones are connected.

Each goal defined gives birth to its opposite goal, which prevents from reaching the main goal. The opposite goal of **Make an eye** is **Remove an eye** , and the opposite goal of **Connect** is **Disconnect**.

These four goals are used in most of the Go programs and their use seems necessary to the development of a rule-based Go playing program.

2.2 A simplified Game Theory

To represent the state of achievement of a goal, I use a simplified Conway Game Theory [Conway 1976], [Berlekamp 1982], [Bouzy 1995].

A game is associated to a goal. It can have different states :

- If the player can reach the goal whatever the opponent plays, the state is '>'.

- If the player can reach the goal if he plays first, but his opponent can prevent him to reach the goal by playing first, the state is '*'.

- If the player cannot reach the goal whatever he plays, the state is '<'.

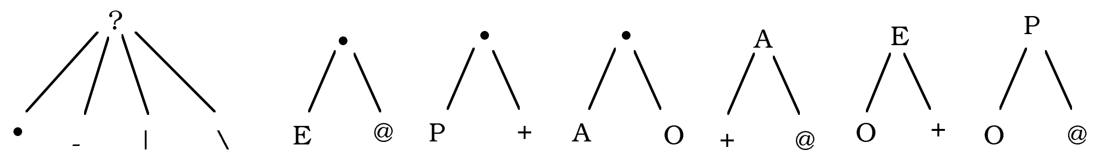
- If the player cannot reach the goal if he plays first, but his opponent cannot prevent him to reach the goal by playing first, the state is '0'.

2.3 Format of the rules

The If-part of rules is a pattern. To construct patterns, Gogol uses various symbols :

@	: Friend stone	O	: Enemy stone
+	: Empty intersection	-	: Exterior of the Goban
|	: Exterior of the Goban	\	: Exterior of the Goban
A	: '@' or '+'	E	: 'O' or '+'
P	: '@' or 'O'	.	: '@' or 'O' or '+'
?	: '@' or 'O' or '+' or '-' or '|' or '\'		

It gives the following hierarchies :



A pattern is included in a two-dimensional rectangle. In a rule they are given as a rectangular matrix of characters, each character in the same line is separated by a space and each line is separated by a carriage return. The key word pattern is placed before the pattern and a ‘!’ determines the end of the pattern.

The then-part is composed of a goal and of a game value as defined previously. If the game is '*', the move allowing to change the value of the game is also given.

A goal is the keyword 'Goal' followed by the name of the goal and by information on the blocks involved.
The end of a rule is determined by a slash.

Example of a rule :

```
pattern
@   @   +
+   +   +
+   @   +
!
Goal Connect block1 0 0 block2 1 2 end
Game >
/
```

The rule says that if the pattern is encountered on the Goban, the block containing the upper left stone is connected to the block containing the lower stone.

3 Automatic Acquisition of Rules

The automatic acquisition of rules is a three step process : Pattern generation, Game evaluation and then Generalisation.

3.1 Pattern generation

Example of a metarule used to generate patterns :

```
.   @   .
.   +   .
.   @   .
```

In this example, each '.' is successively replaced by '@', 'O' and '+'. Therefore 3^6=729 patterns are generated.

It is possible to filter the patterns to be evaluated. A usual filter is to prevent from calculating too specific patterns. The specificity of a pattern is calculated using the elements of the pattern. A '+' counts for 1, a 'O' or a '@' for 4. adding all the counts gives the specificity of the pattern. Gogol usually forgets patterns which have a specificity superior to 40.

3.2 Game evaluation

Gogol makes a list of the goals to calculate on each pattern. If the goal is Connect, he takes all the pairs of friend stones and evaluates the game for each pair, if the goal is Make an eye, he evaluates the game for each intersection not on the edge of the pattern.

For each pattern-goal association, Gogol calculates two MinMax on the pattern. The evaluation function being 1 if the goal is achieved, 0 otherwise. The search ends when no more moves are possible. The first tree is calculated with a friend move first. The second tree is calculated with an opponent move first. Given the two root values, Gogol decide upon the value of the game for the pattern :

Root value Friend plays first	Root value Enemy plays first	Value of the G
1	1	>
1	0	*
0	1	0
0	0	<

When a game is '*', it give birth to sets of rules. One set of rules for the fri
reaching the goal and one set of rules for the enemy moves preventing to reach th

3.3 Generalisation

Generalisation is useful for at least two reasons. The first reason is that it saves time for matching to have the most possible general rules.

Example :

```
pattern              pattern      pattern
@   @   @                 @   @   O              @   @   +
+   +   +                 +   +   +              +   +   +
+   @   +                 +   @   +              +   @   +
!                         !                      !
```

Matching the three previous patterns takes three times longer than matching the following one :

```
pattern
@   @   .
+   +   +
+   @   +
!
```

and they all have the same conclusion :

Goal Connect block1 0 0 block2 1 2 end
Game >

The second reason is that it is easier for the developer of a system to verify a small set of generalised rules than a large set of specific rules.

For each calculated rule, Gogol has two ways to generalise its knowledge. The first way is to match the new rule against all the already learned rules. If the new rule is more general than an already calculated one, the less general is removed from the set of rules. The other way is to generalise using the hierarchies presented in section 2.3. If two rules have the same conclusion and only differ by one element in the pattern, and if the two elements have a common antecedent and if this antecedent has only two branches. It then generalises the new rule by replacing the element by the antecedent and it removes the old rule.

4 The use of the learned rules in Gogol

Gogol is written in C++, it has 40 000 lines of code and uses thirty classes. It participates to the computer Go ladder [Pettersen 1994]

Gogol has learned thousands of tactical Go rules using the method described in this article. Hundred of them are very useful and used in most of the game it plays, others are too specific to be used while playing, the cost in time of matching them is too high compared to the increase in level they give.

4.1 Meaning of the game value in a global context

In this section, I will present some metaknowledge used by Gogol. Metaknowledge is knowledge about knowledge [Pitrat 90]. The metarules described here are rules about the validity in a global context of locally proven rules. To know the validity in a global context of a game calculated locally, goals must be classified in two distinct categories :

- Achievable goals, the goals for which a desired state is known. This goals are Connect and Make an eye.

- Unachievable goals, the goals which prevent from reaching a reachable goal. This goals are Disconnect and Remove an eye.

For achievable goals, the following rules can be applied :

If a game is locally proven '>'
Then it is also globally proven '>'

If a game is locally proven '*'
Then it is also globally proven '*' or '>'

If a game is locally proven '<'
Then it is also globally proven '<' or '*' or '>'

For unachievable goals, the following rules can be applied :

If a game is locally proven '<'
Then it is also globally proven '<'

If a game is locally proven '*'
Then it is also globally proven '*' or '<'

If a game is locally proven '>'
Then it is also globally proven '>' or '*' or '<'

Example :

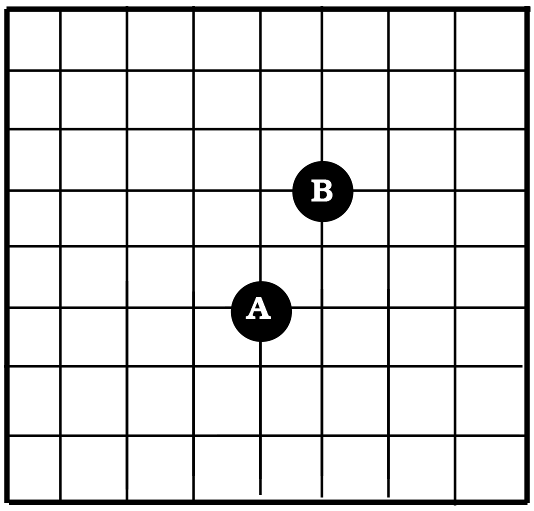


Figure 1

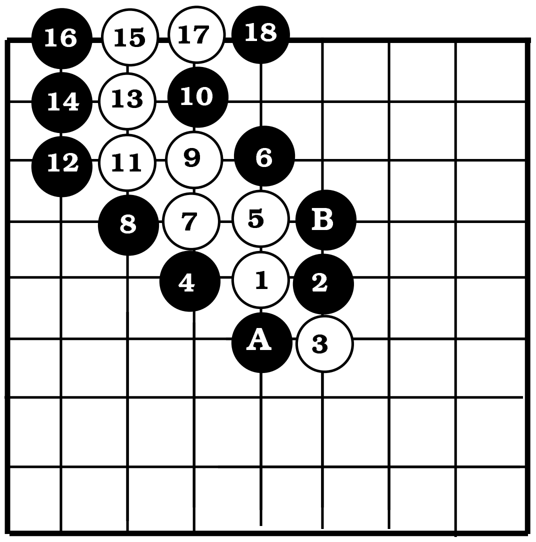


Figure 2

In Figure 1, the following local rule applies :

```
pattern
.   @
+   +
@   +
!
Goal Connect block1 1 0 block2 0 2 end
Game *
Move @ in 1 1
/
```

Hence, the game is locally '*'. However the game can be proved globally '>' as shown in figure 2. As Connect is an achievable goal, this result was predicted by the metarule saying that an achievable goal locally proved '*' can be globally either '*' or '>'. These metarules can be viewed as admissible heuristics upon the game theory.

4.2 Limitations of the learning method

Gogol has a way to represent information on liberties in its rules, the conditions on liberties begin with the key word 'liberty' and end with the keyword 'end'. The described way of learning does not allow Gogol to learn rules containing conditions exterior to the pattern. As an example, he cannot learn with this method the following rule :

```
pattern
@   @
```

```
O   +
@   +
!
liberty 0 1 = 1
Goal Connect block1 0 0 block2 0 2 end
Game *
Move @ in 1 1
/
```

The described method will evaluate the pattern as having a game '<'. This problem will be solved in the next version of Gogol using a type of Explanation Based Learning method.

5 Conclusion

I have shown how to learn many useful tactical rules on eyes and connections. This learning approach allows to find all the rules contained in a specified domain. The tactical rules calculated by the program are rules on eyes and connections. It provides a good starting point for Go programmers desiring to write a rule-based Go program. The interest of the work is to allow to find all the rules corresponding to the specifications given by the programmer, preventing him to enter by hand thousands rules which are necessary to all rule-based programs, and providing a way not to forget anyone.

My main work now is to adapt Explanation Based Learning techniques to Go. I will use databases of problems [Wolf 1994], [Müller 1995] to learn from.

Bibliography

[Berlekamp 1982] - E. Berlekamp, J.H. Conway, R.K. Guy. *Winning Ways*. Academic Press, London 1982.

[Bouzy 1995] - Bruno Bouzy. *Modélisation cognitive du joueur de Go*. Thèse de l'université Paris 6, 1995.

[Brügmann 1993] - Bernd Brügmann. *Monte Carlo Go*. On server ftp.bsdserver.ucsf.edu, 1993.

[Conway 1976] - J. Conway, *On Numbers and Games*, Academic Press, Londres/New-York, 1976.

[Enderton 1991] - Herbert Enderton. *The Golem Go program*. Technical Report Carnegie Mellon University, 1992.

[Enzenberger 1995] - Markus Enzenberger. *Neural networks and Go*. Message sent to the Computer Go mailing list, 30 Jan. 1995.

[Kojima 1994] - Takuya Kojima, Kazuhiro Ueda, Saburo Nagano. *A case study on acquisition and refinement of deductive rules based on EBG in an adversary game : how to capture stones in Go*. Proceedings of the Game Programming Workshop in Japan'94, pp. 34-43.

[Müller 1995] - Martin Müller. *Computer Go as a Sum of Local Games : An Application of Combinatorial Game Theory*. Thesis og the Swiss Federal Institute of Technology Zürich, 1995.

[Pell 1991] - Barney Pell. *Exploratory Learning in the Game of GO*. In D.N.L. Levy and D.F. Beal, editors, Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad. Ellis Horwood, 1991.

[Pettersen 1994] - Eric Pettersen. *The Computer Go Ladder*. Page WWW : http://cgl.ucsf.edu/go/ladder.html, 1994.

[Pitrat 1990] - Jacques Pitrat. *Métaconnaissance futur de l'intelligence artificielle.* Hermès, 1990.

[Schraudolph 1994] - N. Schraudolph, *Temporal Difference Learning of Position Evaluation in the Game of Go*, Neural Information Pocessing Systems 6, Morgan Kaufmann, 1994. Available by ftp bsdserver.ucsf.edu.

[Stoutamire 1991] - D. Stoutamire, *Machine learning, Game Play, and Go*. MS thesis, Case Western Reserve University, 1991.

[Wolf 1994] - T. Wolf, *The program GoTools and its computer-generated tsume-go database*, First Game Programming Workshop in Japan, Hakone, Octobre 1994.

ANNEX

The following rules were created using the metarule :

```
. . .
. . .
. . .
```

and the goal Make an eye. To save place, the format of the rules was a little modified in this listing.

```
pattern
@ @ @
@ O @
@ + .
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ @
@ O @
. + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ @
@ O @
+ + +
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ @
@ O +
@ @ .
!
Eye @ Game *
Move @ 2 1
end
```

```
/
pattern
@ @ @
@ O +
. @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ @
@ O +
+ @ +
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ @
@ + @
@ @ .
!
Eye @ Game >
end
/
pattern
@ @ @
@ + @
@ + .
!
Eye @ Game *
Move @ 1 2
end
```

```
/
pattern
@ @ @
@ + @
O @ +
!
Eye @ Game *
Move @ 2 2
end
/
pattern
@ @ @
@ + @
. @ @
!
Eye @ Game >
end
/
pattern
@ @ @
@ + @
+ @ O
!
Eye @ Game *
Move @ 0 2
end
/
pattern
@ @ @
@ + @
+ @ +
!
Eye @ Game >
end
/
```

```
pattern
@ @ @
@ + @
. + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ @
@ + @
+ + +
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ @
@ + +
@ @ .
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ @
@ + +
. @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ @
@ + +
+ @ +
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ @
+ O @
@ @ .
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ @
+ O @
. @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ @
+ O @
+ @ +
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ @
+ + @
@ @ .
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ @
+ + @
. @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ @
+ + @
+ @ +
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ O
@ + @
@ @ +
!
Eye @ Game *
Move @ 2 2
end
/
pattern
@ @ O
@ + @
+ @ @
!
Eye @ Game *
Move @ 0 2
end
/
pattern
@ @ .
@ O @
@ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ +
@ O @
@ + +
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ +
@ O @
+ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ .
@ O +
@ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ +
@ O +
@ @ +
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ +
@ O +
+ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ .
@ + @
@ @ @
!
Eye @ Game >
end
/
pattern
@ @ +
@ + @
@ @ O
!
Eye @ Game *
Move @ 2 0
end
/
pattern
@ @ +
@ + @
@ @ +
!
Eye @ Game >
end
/
pattern
@ @ .
@ + @
@ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ +
@ + @
@ + +
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ +
@ + @
O @ @
!
Eye @ Game *
Move @ 2 0
end
/
pattern
@ @ +
@ + @
+ @ @
!
Eye @ Game >
end
/
pattern
@ @ +
@ + @
+ @ +
!
Eye @ Game *
Move @ 2 0
end
/
pattern
@ @ +
@ + @
+ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
@ @ .
@ + +
@ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ +
@ + +
@ @ +
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ +
@ + +
+ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
@ @ .
+ O @
@ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ +
+ O @
@ @ +
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ +
+ O @
+ @ @
!
Eye @ Game *
```

```
Move @ 0 1
end
/
pattern
@ @ .
+ + @
@ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ +
+ + @
@ @ +
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ @ +
+ + @
+ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
@ + @
@ O @
@ @ .
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + @
@ O @
. @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + @
@ O @
+ @ +
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + @
@ + @
@ @ .
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + @
@ + @
. @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + @
@ + @
+ @ +
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + .
@ O @
@ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + +
@ O @
@ @ +
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + +
@ O @
+ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + .
@ + @
@ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + +
@ + @
@ @ +
!
Eye @ Game *
Move @ 1 0
end
/
pattern
@ + +
@ + @
+ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
O @ @
@ + @
@ @ +
!
Eye @ Game *
Move @ 2 2
end
/
pattern
O @ @
@ + @
+ @ @
!
Eye @ Game *
Move @ 0 2
end
/
pattern
O @ +
@ + @
@ @ @
!
Eye @ Game *
Move @ 2 0
end
/
pattern
. @ @
@ O @
@ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
+ @ @
@ O @
@ + +
!
Eye @ Game *
Move @ 1 2
end
/
pattern
+ @ @
@ O @
+ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
. @ @
@ O +
@ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
+ @ @
@ O +
@ @ +
!
Eye @ Game *
Move @ 2 1
end
/
pattern
+ @ @
@ O +
+ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
. @ @
@ + @
@ @ @
!
Eye @ Game >
end
/
pattern
+ @ @
@ + @
@ @ O
!
Eye @ Game *
Move @ 0 0
end
/
pattern
+ @ @
@ + @
@ @ +
!
Eye @ Game >
end
/
pattern
. @ @
@ + @
@ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
+ @ @
@ + @
@ + +
!
Eye @ Game *
Move @ 1 2
end
/
pattern
+ @ @
@ + @
O @ @
!
Eye @ Game *
Move @ 0 0
end
/
pattern
+ @ @
@ + @
+ @ @
!
Eye @ Game >
end
/
pattern
+ @ @
@ + @
+ @ +
!
Eye @ Game *
Move @ 0 0
end
/
pattern
+ @ @
@ + @
+ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
. @ @
@ + +
@ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
+ @ @
@ + +
@ @ +
!
Eye @ Game *
Move @ 2 1
end
/
pattern
+ @ @
@ + +
+ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
. @ @
+ O @
@ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
+ @ @
+ O @
@ @ +
!
Eye @ Game *
Move @ 0 1
end
/
pattern
+ @ @
+ O @
+ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
. @ @
+ + @
@ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
+ @ @
+ + @
@ @ +
!
Eye @ Game *
Move @ 0 1
end
/
pattern
+ @ @
+ + @
+ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
+ @ O
@ + @
@ @ @
!
Eye @ Game *
Move @ 0 0
end
/
pattern
+ @ +
@ O @
@ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
+ @ +
@ O +
@ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
+ @ +
@ + @
@ @ @
!
Eye @ Game >
end
/
pattern
+ @ +
@ + @
@ @ +
!
Eye @ Game *
Move @ 0 0
end
/
pattern
+ @ +
@ + @
@ + @
!
Eye @ Game *
Move @ 1 2
end
/
pattern
+ @ +
@ + @
+ @ @
!
Eye @ Game *
Move @ 0 0
end
/
pattern
+ @ +
@ + +
@ @ @
!
Eye @ Game *
Move @ 2 1
end
/
pattern
+ @ +
+ O @
@ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
+ @ +
+ + @
@ @ @
!
Eye @ Game *
Move @ 0 1
end
/
pattern
. + @
@ O @
@ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
+ + @
@ O @
@ @ +
!
Eye @ Game *
Move @ 1 0
end
/
pattern
+ + @
@ O @
+ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
. + @
@ + @
@ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
+ + @
@ + @
@ @ +
!
Eye @ Game *
Move @ 1 0
end
/
pattern
+ + @
@ + @
+ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
+ + +
@ O @
@ @ @
!
Eye @ Game *
Move @ 1 0
end
/
pattern
+ + +
@ + @
@ @ @
!
Eye @ Game *
Move @ 1 0
end
/
```