A Phantom Go Program

Tristan Cazenave

Labo IA, Université Paris 8, 2 rue de la Liberté, 93526, St-Denis, France
cazenave@ai.univ-paris8.fr

Abstract. This paper presents a Phantom Go program. It is based on a Monte-Carlo approach. The program plays Phantom Go at an intermediate level.

1 Introduction

Phantom Go is often played at Go congress and provides enjoyable games both for the players and the surrounding public. Phantom Go is a game with hidden information. Monte-Carlo methods work well in Go and in games of incomplete information, and as Phantom Go is Go with incomplete information, Monte-Carlo methods are expected to work well in Phantom Go. This paper demonstrates that it is the case and shows the results of a Monte-Carlo based program that plays Phantom Go.

In the second section, we present the game of Phantom Go. In the third section, we recall previous work on Monte-Carlo Go. In the fourth section, we detail how Monte-Carlo is adapted to Phantom Go. In the fifth section, we give experimental results. The last section outlines future work and concludes.

2 Phantom Go

Phantom Go is a two player game. There are two players and a referee. There are three boards, one for each player and one for the referee, the board of the referee is called the reference board. It is usually played on 9x9 boards. The referee can see all three boards. Each player can only see his own board. When it is a player's turn, he chooses a move and asks the referee if it is legal for him to play on the intersection of this move by pointing at the intersection. The referee answers 'legal move' or 'illegal move' according to the reference board. If the move is illegal, the player chooses another move, and so on until he finds a legal move. When the player finds a legal move, it is played on the reference board by the referee, and by the player on his own board. If there is a capture, the referee announces the number of stones being captured and tells the other player which stones have been captured. After a move has been played, it is the other player's turn. The game ends when both players pass.

Phantom Go is the equivalent for Go of Kriegspiel for Chess [1].

3 Monte-Carlo Go

Monte-Carlo methods compute statistics on more or less random games in order to find the best move. They have been used in games such as Bridge [2], Poker [3], Tarok [4]

and Scrabble [5] for example. All this games have hidden information which make them particularly suited for Monte-Carlo. However, Monte-Carlo methods have also proven useful in complete information games, and particularly in Go. Bruegmann [6] was the first to experiment Monte-Carlo in Go. Recently, other Go programs have started using it, and improved it, simplifying the method and proposing basic improvements [7], combining it with a knowledge based program that selects a few number of moves that are evaluated by the Monte-Carlo method [8] or combining it with tactical search [9].

There are several slightly different ways to write a Monte-Carlo Go program [7]. In this paper, we use as Monte-Carlo Go the following algorithm: the program plays a large number (usually 1,000 to 10,000) of random games starting at the current position. The moves of the random games are chosen almost randomly among the legal moves, except that they must not fill the player's eyes. A player passes in a random game when his only legal moves are on his own eyes. The game ends when both players pass. In the end of each random game, the score of the game is computed using Chinese rules (in our case, it consists in counting one point for each stone and each eye of the player's color, and subtracting the opponent count from the player's count). The program computes, for each intersection, the mean results of the random games where it has been played first by one player, and the mean for the other player. The value of a move is the difference between the two means. The program plays the move with the highest value.

4 Monte-Carlo Phantom Go

Monte-Carlo Go plays a game of complete information. Monte-Carlo Phantom Go has to deal with hidden information. In order to cope with this hidden information, the program has to guess where the stones of the opponent are, and the best move on average against different repartitions of the opponent stones.

The basic Monte-Carlo Go method is reused: the program plays many games randomly with the constraint of not filling its own eyes. However, all the random games do not start with the same position, as the program does not know exactly the real position. The program memorizes all the forbidden moves. It places an opponent stone on each of the forbidden moves. The program also knows the number of opponent stones that are present on the reference board. Substracting the number of forbidden moves from the number of opponent stones gives the number of stones with an unknown position.

At the beginning of each random game, after it has placed its stones, and opponent stones on the forbidden intersections, it randomly places opponent stones on the empty intersections left. It randomly places as many opponent stones as there are opponent stones with an unknown position. Once all the stones are placed, it plays a random game starting with a move of its color, and playing moves randomly on empty intersections, provided they are not a player's eye.

A player passes when his only moves left are his own eyes. When both players pass, the game is ended. At the end of a random game, the score is computed using chinese rules. For each move played during the random game, the mean score of playing this move for all the random games is updated to take into account the result of the game. When the 10,000 random games have been played, the program substracts, for all the

legal moves, the mean score of the move for the opponent's color from the mean score of the move for the player's color. The move that has the highest difference is tried.

If the move is told to be illegal by the referee, the program memorizes it as a forbidden move, and starts its process again. It plays 10,000 new random games, taking into account the new information given by the referee on the forbidden move.

5 Experimental Results

The number of random games played at each move is set to 10,000. The program plays a move in a few seconds on a Pentium 3.0 GHz. The first subsection details an example games. The second subsection gives results against different Go players.

5.1 An example game

The author is an european one dan Go player. Playing games with the program usually results in a small win by the author. An example 9x9 game is given below. When a player tries an illegal move, it is reported in the game's notation, and therefore multiple moves by the same player follow each other. All but the last move by the same player are illegal, they are given because they give information on the knowledge of the game by the player and are required to analyze and understand the game. The author is white and the program is black.

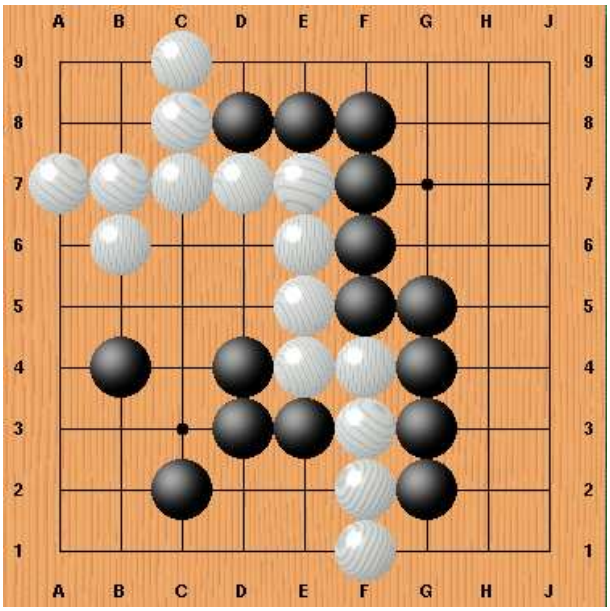

Fig. 1. The reference board after 30 moves

B(D4); W(E5); B(E3); W(E6); B(F5); W(E4); B(E6); B(F6); W(E7); B(F7); W(E3); W(D4); W(F4); B(E5); B(F4); B(E4); B(G4); W(F3); B(F3); B(E7); B(E8); W(F2); B(G3); W(F1); B(F2); B(D8); W(E8); W(D7); B(F8); W(D8); W(C7); B(G2); W(C8); B(G5); W(C9); B(D3); W(F7); W(D3); W(B7); B(C7); B(D7); B(C8); B(F1); B(C2); W(A7); B(B4); W(G4); W(G3); W(G2); W(B6); B(B6); B(B3); W(B5); B(C9); B(B5); B(E2); W(B4); W(A5); B(C5); W(A4); B(B8); W(A3); B(B7); B(H6); W(B3); W(A2); B(E1); W(B2); B(A8); W(C2); W(B1); B(G1); W(C1); B(D5); W(D1); B(A5); B(A7); B(H7); W(F6); W(F5); W(D9); B(C3); W(E9); B(B9); W(F8); W(F9); B(G8); W(G9); B(A4); B(A3); B(A2); B(C1); B(D1); B(B1); B(B2); B(D9); B(E9); B(F9); B(G9); B(H9); W(G8); W(H9); W(H8); B(H8); B(J8); W(J9); W(J8); W(J7); B(J7); B(H5); W(H8); W(G1); W(H7); W(H6); W(Pass); B(J6);

In this game, the author has the strategy of dividing the board in two, and killing one side of the board after it has been divided. This strategy is used by some experienced Phantom Go players and it is quite efficient.

Figure 1 gives the reference board after the first thirty moves of the game.

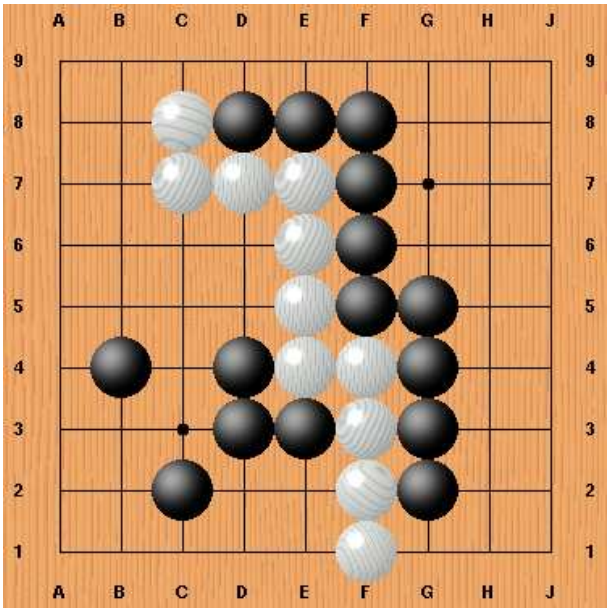

Fig. 2. The program board after 30 moves

Figure 2 gives the program board after the first thirty moves of the game. In this position, the program has guessed most of the white stones, and is ahead of white. It has to close the borders of its right group, and to make two eyes with its left group.

Figure 3 gives the reference board at the end of the game. The program failed to make its left group live and lost the game.

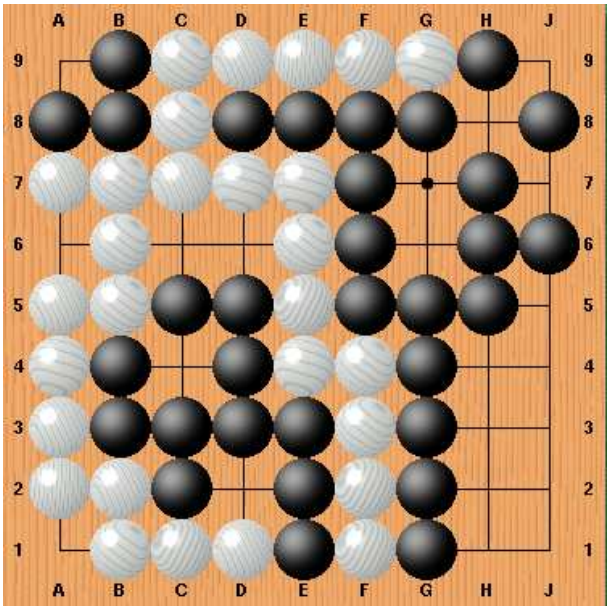

Fig. 3. The final position

5.2 Results against Go players

In order to better test the phantom Go program, it has played 9x9 games against Go players of different levels. The results can be seen in table 1. For each game we give the level of the human player, the white player, the black player and the result.

Table 1. Results of the program against Go players.

level	*white*	*black*	*result*
13 kyu	nicolas	program	W+8.5
13 kyu	program	nicolas	W+resign
5 dan	bernard	program	W+13.5
13 kyu	program	arpad	W+17.5
5 dan	program	bernard	W+27.5
5 dan	bernard	program	W+47.5

As the results show, the program can win by 27.5 points against a 5 dan Go player as well as lose by 8.5 points against a 13 kyu Go player. We can conclude that the program has the level of experienced Go players who have only played a few number of phantom

Go games. We do not know experienced (and even less ranked) phantom Go players to test the program against.

6 Conclusion and future work

We have presented a Phantom Go program based on a Monte-Carlo approach. It plays interesting Phantom Go games. The particularity of the application of Monte-Carlo to Phantom Go is that unknown stones are placed at random at the beginning of each random game.

A possible improvement is to deal with usual Phantom Go strategies in the random games. For example the divide and kill strategy used by the author can be used in some random games to bias the move selection.

Other possible improvements are the improvements used in Monte-Carlo Go programs. For example, it is possible to combine with tactical search by computing the results of simple tactical searches at the beginning of the random games, so as to compute statistics in the random games on the associated goal. Once the statistics on the goals are computed they give a better evaluation of the associated move than basic statistics on moves. Such an approach has worked in Monte-Carlo Go [9], and could well work too in Phantom Go. Another possible improvement is to use patterns to bias the selection of moves in the random games so as to improve their quality as in [8].

The program also has problems dealing with semeais, and opponent eyes. It can be improved with specialized knowledge.

References

1. Bolognesi, A., Ciancarini, P.: Computer programming of Kriegspiel endings: The case of KR versus K. In: Advances in Computer Games 10, Graz, Austria, Kluwer (2003) 325–342
2. Ginsberg, M.L.: GIB: Steps toward an expert-level bridge-playing program. In: IJCAI-99, Stockholm, Sweden (1999) 584–589
3. Billings, D., Davidson, A., Schaeffer, J., Szafron, D.: The challenge of poker. Artificial Intelligence **134** (2002) 210–240
4. Lustrek, M., Gams, M., Bratko, I.: A program for playing tarok. ICGA Journal **26** (2003) 190–197
5. Sheppard, B.: Efficient control of selective simulations. ICGA Journal **27** (2004) 67–80
6. Bruegmann, B.: Monte Carlo Go. ftp://ftp-igs.joyjoy.net/go/computer/mcgo.tex.z (1993)
7. Bouzy, B., Helmstetter, B.: Monte Carlo Go developments. In: Advances in computer games 10, Kluwer (2003) 159–174
8. Bouzy, B.: Associating domain-dependent knowledge and monte carlo approaches within a go program. In: Joint Conference on Information Sciences, Cary (2003) 505–508
9. Cazenave, T., Helmstetter, B.: Combining tactical search and monte-carlo in the game of go. In: CIG'05, Colchester, UK (2005)