Intelligent Robotics 3

Homework- Go game

By: Mohak Patel

DESCRIPTION:

Go is an abstract strategy board game for two players, in which the aim is to surround more territory than the opponent.

The game was invented in China more than 2,500 years ago and is believed to be the oldest board game continuously played today. It was considered one of the four essential arts of the cultured aristocratic Chinese scholars in antiquity. The earliest written reference to the game is generally recognized as the historical annal Zuo Zhuan .

Rules of GO:

A game of Go starts with an empty board. Each player has an effectively unlimited supply of pieces (called **stones**), one taking the black stones, the other taking white. The main object of the game is to use your stones to form territories by surrounding vacant areas of the board. It is also possible to capture your opponent's stones by completely surrounding them.

Players take turns, placing one of their stones on a vacant point at each turn, with Black playing first. Note that stones are placed on the intersections of the lines rather than in the squares and once played stones are not moved. However they may be captured, in which case they are removed from the board, and kept by the capturing player as **prisoners**.

At the end of the game, the players count one point for each vacant point inside their own territory, and one point for every stone they have captured. The player with the larger total of territory plus prisoners is the winner.

Steps:

1. The board is empty at the onset of the game (unless players agree to place a handicap).
2. Black makes the first move, after which White and Black alternate.
3. A move consists of placing one stone of one's own color on an empty intersection on the board.
4. A player may pass their turn at any time.
5. A stone or solidly connected group of stones of one color is captured and removed from the board when all the intersections directly adjacent to it are occupied by the enemy. (Capture of the enemy takes precedence over self-capture.)
6. No stone may be played so as to recreate a former board position.
7. Two consecutive passes end the game.
8. A player's area consists of all the points the player has either occupied or surrounded.
9. The player with more area wins.

Diagram 1 (a) shows the position at the end of a game on a 9 by 9 board, during which Black captured one white stone at **a**.

Black has surrounded 15 points of territory, 10 in the lower right corner and 5 towards the top of the board. Black's territory includes the point **a** formerly occupied by the white stone Black has captured. Adding this prisoner, Black has a total of 16 points. White's territory is 17 points, so White wins the game by one point.

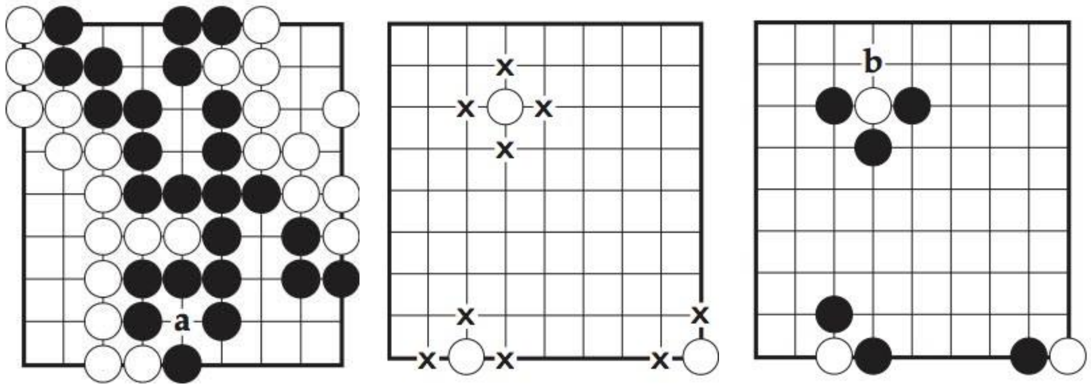


Diagram 1.

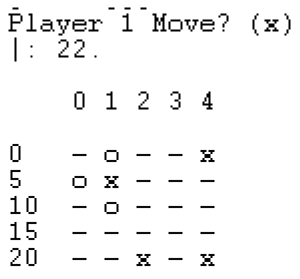
Diagram 1(b) shows three isolated white stones with their liberties marked by crosses. Stones which are on the edge of the board have fewer liberties than those in the center of the board. A single stone on the side has three liberties, and a stone in the corner has only two liberties.

Diagram 3(c) shows the same three stones of **Diagram 1(b)** each with only one liberty left and therefore subject to capture on Black's next turn. Each of these white stones is said to be in atari, meaning they are about to be captured.

Implementation of Go in Prolog and Results:

This prolog code show the implantation of the 5x5 Go game. Start the game by typing (play.). To enter the position of the move, add the first row number (0,1,2,3,4) to the first column number (0,5,10,15,20).

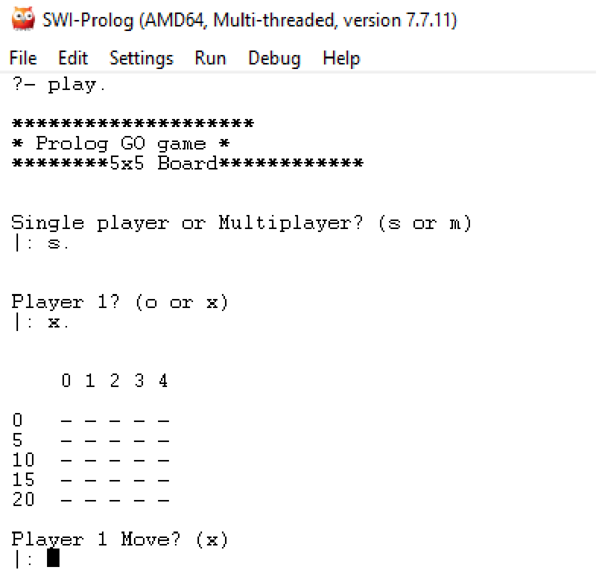
Example: Here player (x) place at position 22 (row(2)+column(20))



```
Player 1 Move? (x)
|: 22.

     0 1 2 3 4

0    - o - - x
5    o x - - -
10   - o - - -
15   - - - - -
20   - - x - x
```

For starting game:

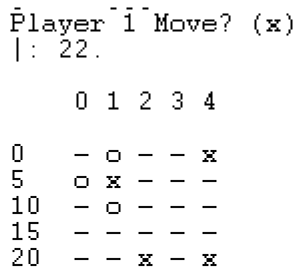
1. Play.
2. Choose single player (Human Vs Computer) or Multiplayer (Human Vs Human)
3. Black Player will start the game.


SWI-Prolog (AMD64, Multi-threaded, version 7.7.11)
File Edit Settings Run Debug Help

```
?- play.

*********************
* Prolog GO game *
*********5x5 Board*************


Single player or Multiplayer? (s or m)
|: s.


Player 1? (o or x)
|: x.


    0 1 2 3 4

0   - - - - -
5   - - - - -
10  - - - - -
15  - - - - -
20  - - - - -

Player 1 Move? (x)
|: 
```

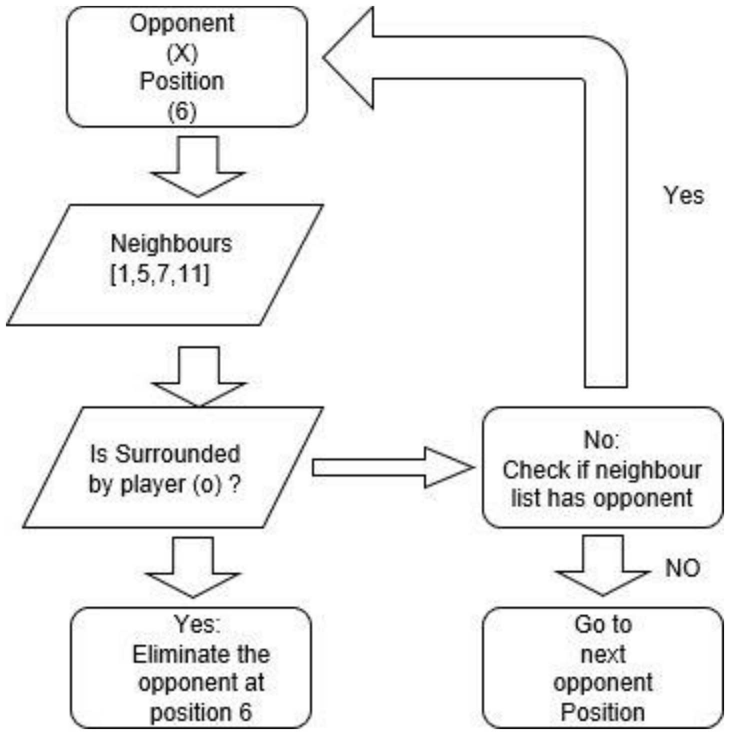
Capturing Algorithm:

For capturing the opponents and to detect the captured opponents, I have used tree expansion and searching the opponents/player at nodes of tree. It uses the prolog backtracking to find all solutions. For example, as shown in below figure, player (x) is almost surrounded by player (o). So in the next turn of player (o), it will place at position 7 so that it can eliminate the one stone of player (x) placed at position 6.



```
Player 1 Move? (x)
|: 22.

    0 1 2 3 4

0   - o - - x
5   o x - - -
10  - o - - -
15  - - - - -
20  - - x - x
```

Flowchart:

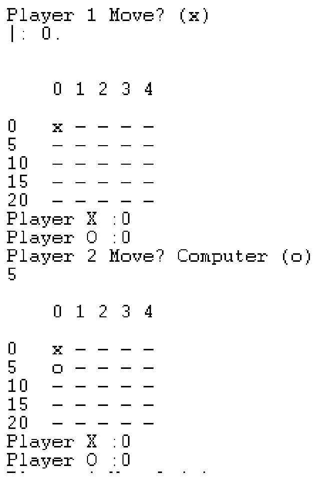


Opponent (X) Position (6)
Neighbours [1,5,7,11]
Is Surrounded by player (o) ?
Yes: Eliminate the opponent at position 6
No: Check if neighbour list has opponent
Yes
NO
Go to next opponent Position

Human Vs Computer (AI):

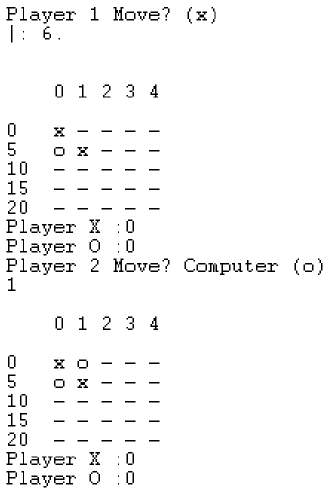
I have used same capturing algorithm to predict the computer move by assuming that opponent is almost surrounded by player and only one move is left to capture the opponent. First computer will play one prediction move and use capturing algorithm. If prediction move gives the output yes of capturing algorithm then it will be the best move for computer to play. And if it will give output no then algorithm will first find the opponent who has minimum neighbours so that computer can capture that opponent by playing minimum moves and then it will play at any neighbour position of that opponent.

For example, in below figure 1, player played at position (0). So computer played at position (5) to capture opponent. In figure 2, player played at position (6). So computer played at position (1) to capture the opponent. Figure 3 shows the result of that



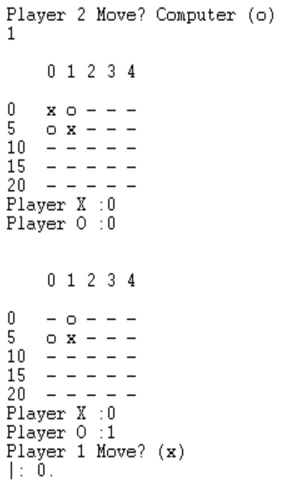
```
Player 1 Move? (x)
|: 0.

    0 1 2 3 4

0   x - - - -
5   - - - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
Player 2 Move? Computer (o)
5

    0 1 2 3 4

0   x - - - -
5   o - - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
```

Figure 1



```
Player 1 Move? (x)
|: 6.

    0 1 2 3 4

0   x - - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
Player 2 Move? Computer (o)
1

    0 1 2 3 4

0   x o - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0
```

Figure 2



```
Player 2 Move? Computer (o)
1

    0 1 2 3 4

0   x o - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :0


    0 1 2 3 4

0   - o - - -
5   o x - - -
10  - - - - -
15  - - - - -
20  - - - - -
Player X :0
Player O :1
Player 1 Move? (x)
|: 0.
```

Figure 3

<u>Troubleshooting:</u>

One the main advantage of prolog is backtracking but this is 450 lines of prolog code and sometimes prolog is giving unnecessary output so it is very difficult to find the place from where I can stop the backtracking.

<u>Conclusion:</u>

This Go game works mostly similar to the original Go game rules. The capturing algorithm (Chain rule) is working correctly to find captured opponents. The prolog code is for 5x5 Go game but code is made in such way that it can be modify for NxN Board and capturing algorithm will work correctly. In this GO game there are infinite numbers of winning moves so it will be very difficult to define winning moves like tic-tac-toe game. This prolog Go game uses one predicting algorithm. It is suitable for just smaller board size. It will not work very well for bigger board size.