The PracTEX Journal, 2008, No. 2
Article revision 2008/8/25

Go Game Positions with METAPOST

Wentao Zheng

Email zhengwt@cn.ibm.com
Address IBM China Research Laboratory

Abstract This article introduces a method of drawing Go game positions with METAPOST. It begins with how the Go game is modeled in the METAPOST language, then explains the detailed implementation, and ends with some examples of Go game positions.

1 Introduction

Go is a board game that originated in ancient China in the 4th century BC. In Chinese, it is known as *weiqi*, which means "board game of surrounding". It is commonly known in the West by its Japanese name *igo* (the Japanese reading of its Chinese name), because early western players learned the game from Japanese sources.

Go is considered the most complex game of all, and has a long history of academic study. Often the Go game graphics in research papers are drawn with low quality. This article introduces a method of drawing Go game positions with METAPOST that can produce flexible and high quality Go game graphics.

This article is organized as follows: section 2 introduces modelling the Go game in METAPOST; section 3 describes the detailed implementation; and section 4 shows some examples of using this method to draw Go game graphics.

2 Modeling Go in METAPOST

Go is played by two players alternately placing black and white stones on the vacant intersections of a 19×19 grid board (see Figure 1). The object of the game is to control a larger part of the board than the opponent. In a game position, if there are no adjacent intersections for a stone or a group of stones, it is considered

as captured and the stones are removed from the board. For more information about Go game rules, please see 'Go (board game)' from Wikipedia [2].

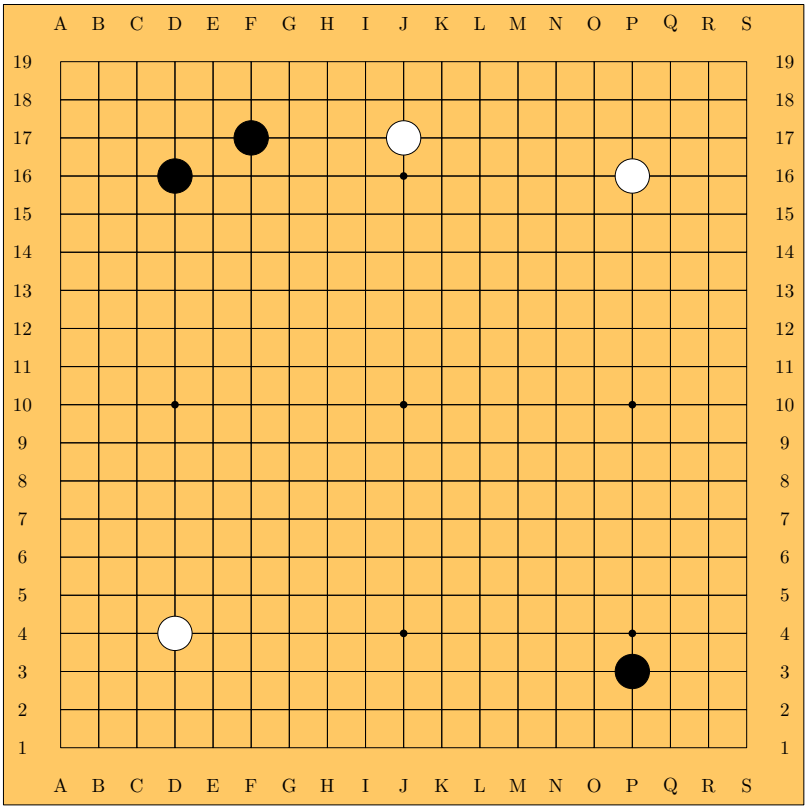


Figure 1: A standard Go game board with stones

In the initial Go game position, there are no stones. To begin a game, the first player puts a black stone, and then the opponent puts a white stone. These two steps are repeated until no more stones can be placed. On any turn a player may choose to skip his or her play. From a programming perspective, this process can be described as follows:

1. Initialize the Go board
2. Black player puts a stone (optional)
3. White player puts a stone (optional)
4. Repeat steps 2 and 3 until no stones can be put on board

In each stone-putting action, we check whether the stone will capture the opponent's stones or be captured (suicide). We model a Go game position in METAPOST as four major routines (components)

- Board initialization
- Stone put
- Capture check
- Board display

We will describe each component in detail in the next section.

3 Implementation

3.1 Board initialization

To provide a flexible way to initialize the board, we use several internal parameters (Figure 2). These parameters control how a Go board will be rendered.

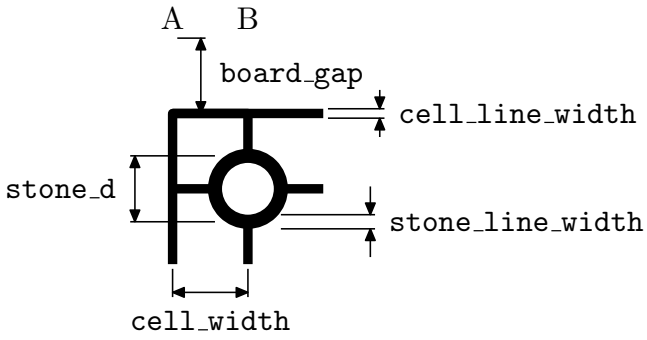


Figure 2: Go board parameters.

board_color the background color of the Go board

cell_width the width of the board grid

cell_line_width the width of line used in drawing the board

stone_d the diameter of the stone (default as $0.9 \times$ cell_width)

stone_line_width the width of line used in drawing the stone outline

board_gap the gap between board edges and board labels

board_size the size of the board (default as 19)

The board status is recorded in METAPOST as a two-dimensional array

```
numeric mem[][]
```

Each element in the array represents a Go board intersection, and the value can be

-1 white stone

1 black stone

0 empty

There is a routine called

```
init_board
```

that is used to initialize a Go board with all intersections being empty.

3.2 The Stone-put

Stones are put on the board by specifying their coordinates. Each coordinate has two values, horizontal and vertical, as shown in Figure 1. The horizontal coordinate is a character starting from the letter 'A', and the vertical coordinate is an integer number starting from 1. If the value is less than 1 or 'A' or greater than `board_size`, it is considered as invalid.

There are two routines for putting stones

put_b put a black stone at specified coordinate

put_w put a white stone at specified coordinate

The coordinates used to identify the positions of stones are useful for human players. But METAPOST does not know where to put the stones directly based on player-oriented coordinates, such as 'A5'. The secret behind this action is a transformation that converts them into METAPOST-understandable coordinates, such as `(20pt,80pt)`. The latter coordinate form is used in the board graphics generation routine.

3.3 Capture check

After putting a stone on the board, it is necessary to check whether the stone captures the opponent's stones or can be captured (suicide). Generally, there are three rules:

- If the stone-put is not a suicide and captures some of the opponent's stones, then the captured stones are removed from board. (Figure 3(a))
- If the stone-put is a suicide and captures none of the opponent's stones, then the stone is removed from board. (Figure 3(b))
- If the stone-put is a suicide and captures some of the opponent's stones, then the stone is reserved and the captured stones are removed from board. (Figure 3(c))

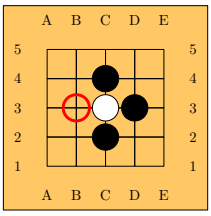
After each move, there are two steps to check for captured stones. In the first step, the routine

```
_capture <coord>
```

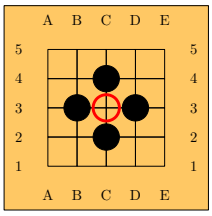
is used to check whether the stone at the specified coordinate will capture the opponent's stones. If so, the captured stones are removed immediately. In the second step, the routine

```
_suicide <coord>
```

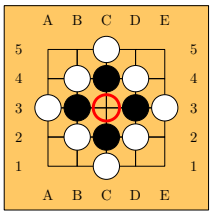
is used to check whether the stone at a specified coordinate is a suicide. If so, the stone is removed. By checking a move in this order, we can ensure that a stone which is both a suicide and captures other stones is reserved and that those captured stones are removed.



(a) Capture (put a black stone)



(b) Suicide (put a white stone)



(c) Capture and suicide (put a white stone)

Figure 3: Three rules for capture check (red circles to put stones)

3.4 Board display

To graphically display a Go game (including board and stones over a sequence of moves), there are two different approaches. The first one is **programmatic**. Go game graphics are generated by calling the routine

```
display_board
```

in METAPOST code. The second approach is **script-driven**. You write a script file which contains information about where to place the stones step by step. By calling the routine

```
script <file_name>
```

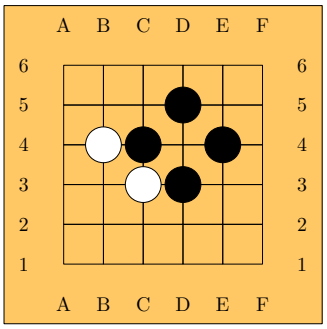
Go game graphics will be generated automatically for each step. In the script file, stone-put operations are separated into lines. The action is formatted as three parts: color, horizontal coordinate, and vertical coordinate. For example, "B A 2" means put a black stone at position (A,2).

4 Examples

In this section, two examples of using our method to generate high quality Go game graphics are presented.

The first one uses the programmatic approach.

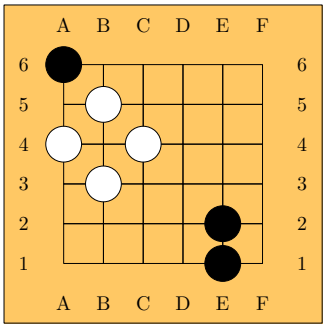
```
beginfig(1);
  set_board_size 6;
  init_board;
  put_b(C4);
  put_w(D4);
  put_b(D3);
  put_w(C3);
  put_b(E4);
  put_w(B4);
  put_b(D5);
  display_board;
endfig;
```



This second example uses the script-driven approach.

```
%%% content in script file "test.go"
B A 6
W B 5
B B 4
W A 4
B E 2
W B 3
B E 1
W C 4

%%% content in MP file "test.mp
set_board_size 6; script "test.go";
```



5 Summary

This article introduced Go game positions with METAPOST. Motivation, general design and implementation of the method were described in separate sections. Additionally, some examples of how to use the method to generate high quality Go game graphics were shown. There are still some points to improve, such as supporting *ko* rule [2].

Acknowledgments

I'd like to thank all those who helped me on writing this article. They are Lance Carnes, Yuri Robbers and two anonymous reviewers.

References

[1] John Hobby, "METAPOST: A User's Manual".

[2] Go (board game), http://en.wikipedia.org/wiki/Go_(board_game)