A Networked *Gothello* Referee: Specification

Bart Massey

November 17, 2019

Gothello is a game of skill created by the author for educational purposes. It is played on an ordinary checkerboard, and has something of the feel of Othello or Go. In this document, we will describe a networked server to which human and computer Gothello players can connect to play the game in a refereed fashion.

1 The Game Of *Gothello*

The rules of Gothello are intended to capture some of the feel of Go, while being more amenable to adversary search. Gothello is a two player game, played by players conventionally designated as *black* and *white*.

$$PLAYER ::= black \mid white$$

$$\begin{array}{|l}
opponent : PLAYER \rightarrowtail\!\!\!\!\twoheadrightarrow PLAYER \\
\hline
opponent\ white = black \\
opponent\ black = white
\end{array}$$

The board, shown in figure 1, is an ordinary checkerboard or go board: black and white stones are placed on intersections (conventionally designated using standard algebraic notation). For the purposes of this document, the game will be played with a 5x5 array of intersections.

$$\begin{array}{l}
DIGIT == 1 \mathrel{..} 5 \\
SQUARE == DIGIT \times DIGIT
\end{array}$$

At any given point in the game, the board position can be given by noting whether each square is blank or contains a colored stone.

$$\begin{array}{l}
SQUAREVAL ::= stone\langle\!\langle PLAYER \rangle\!\rangle \mid blank \\
BOARD == SQUARE \rightarrow SQUAREVAL
\end{array}$$

$$\begin{array}{|l|}
\hline
\quad GothelloPosition \\
board : BOARD \\
to\_move : PLAYER \\
\hline
\end{array}$$

The board begins empty, and black moves first.

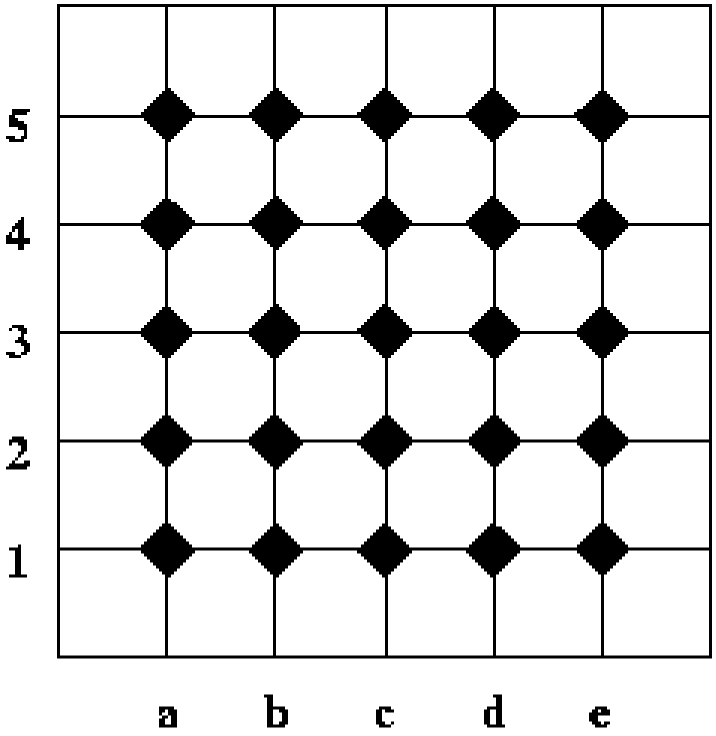

Figure 1: Gothello Board In Initial Configuration

$$\begin{array}{|l}
\hline
\textit{InitGothelloPosition} \\
\textit{GothelloPosition}' \\
\hline
\textit{board}' = \textit{SQUARE} \times \{\textit{blank}\} \\
\textit{to\_move}' = \textit{black} \\
\hline
\end{array}$$

The players alternate in placing stones of their own color on blank spaces on the board. Stones of the same color which are connected horizontally and/or vertically are *neighbors*.

relation (_ $\mathsf{adjoins}$ _)

$$\begin{array}{|l}
\_ \ \mathsf{adjoins} \ \_ : \textit{SQUARE} \leftrightarrow \textit{SQUARE} \\
\hline
(\_ \ \mathsf{adjoins} \ \_) = \\
\qquad \{d, d_1, d_2 : \textit{DIGIT} \mid d_2 = d_1 + 1 \bullet ((d_1, d), (d_2, d))\} \cup \\
\qquad \{d, d_1, d_2 : \textit{DIGIT} \mid d_2 = d_1 + 1 \bullet ((d, d_1), (d, d_2))\}
\end{array}$$

$$\begin{array}{|l}
\textit{neighbor} : \textit{BOARD} \rightarrow (\textit{SQUARE} \leftrightarrow \textit{SQUARE}) \\
\hline
\forall \textit{board} : \textit{BOARD} \bullet \\
\qquad \textit{neighbor board} = \{s_1, s_2 : \textit{SQUARE} \mid s_1 \ \mathsf{adjoins} \ s_2 \wedge \\
\qquad\qquad \textit{board } s_1 \neq \textit{blank} \wedge \textit{board } s_1 = \textit{board } s_2\}
\end{array}$$

A maximal set of mutual neighbors is a *group*. It is most useful to talk about the group containing some specific board position in a given board.

$$
\begin{array}{|l}
\mathit{group} : \mathit{BOARD} \to \mathit{SQUARE} \to \mathbb{P}\,\mathit{SQUARE} \\
\hline
\forall\, \mathit{board} : \mathit{BOARD};\ s : \mathit{SQUARE} \mid \mathit{board}\ s \neq \mathit{blank} \bullet \\
\qquad \mathit{group}\ \mathit{board}\ s = (\mathit{neighbor}\ \mathit{board})^{+} (\!|\, \{ s \} \,|\!)
\end{array}
$$

The other key concept in the rules is the concept of liberties of a group. These are the blank squares immediately surrounding the group. The liberties of a position are the liberties of the group at that position.

$$
\begin{array}{|l}
\mathit{adjoining\_squares} : \mathbb{P}\,\mathit{SQUARE} \to \mathbb{P}\,\mathit{SQUARE} \\
\mathit{group\_liberties} : \mathit{BOARD} \to \mathbb{P}\,\mathit{SQUARE} \to \mathbb{P}\,\mathit{SQUARE} \\
\mathit{liberties} : \mathit{BOARD} \to \mathit{SQUARE} \to \mathbb{P}\,\mathit{SQUARE} \\
\hline
\forall\, \mathit{ss} : \mathbb{P}\,\mathit{SQUARE} \bullet \\
\qquad \mathit{adjoining\_squares}\ \mathit{ss} = (\_ \ \mathsf{adjoins}\ \_)(\!|\, \mathit{ss} \,|\!) \setminus \mathit{ss} \\
\forall\, \mathit{board} : \mathit{BOARD};\ \mathit{ss} : \mathbb{P}\,\mathit{SQUARE} \bullet \\
\qquad \mathit{group\_liberties}\ \mathit{board}\ \mathit{ss} = \mathit{adjoining\_squares}\ \mathit{ss} \cap \mathit{board}^{\sim} (\!|\, \{ \mathit{blank} \} \,|\!) \\
\forall\, \mathit{board} : \mathit{BOARD} \bullet \\
\qquad \mathit{liberties}\ \mathit{board} = (\mathit{group\_liberties}\ \mathit{board}) \circ (\mathit{group}\ \mathit{board})
\end{array}
$$

There are a number of possible outcomes of a player's turn.

$$\mathit{RESULT} ::= \mathit{win}\langle\!\langle \mathit{PLAYER} \rangle\!\rangle \mid \mathit{draw} \mid \mathit{not\_done} \mid \mathit{illegal\_move}$$

A player may pass on any turn, and must pass if no legal move is available. The game is over when both players have passed in succession. This is a function of the immediate history of the game (*i.e.*, the sequence of moves that have been made recently).

$$
\begin{array}{l}
\mathit{MOVE} ::= \mathit{move}\langle\!\langle \mathit{SQUARE} \rangle\!\rangle \mid \mathit{pass} \\
\mathit{HISTORY} == \mathrm{seq}\,\mathit{MOVE}
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\text{— } \mathit{GothelloGame} \text{ —}} \\
\mathit{GothelloPosition} \\
\mathit{history} : \mathit{HISTORY} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\text{— } \mathit{InitGothelloGame} \text{ —}} \\
\mathit{InitGothelloPosition} \\
\mathit{history}' : \mathit{HISTORY} \\
\hline
\mathit{history}' = \varnothing \\
\hline
\end{array}
$$

In short, when the Gothello referee receives an action, it gets a move, records it in the history, and report the result.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{GothelloAction}} \\ \hline
\Delta \textit{GothelloGame} \\
\textit{move?} : \textit{MOVE} \\
\textit{result!} : \textit{RESULT} \\ \hline
\textit{history}' = \langle \textit{move?} \rangle \frown \textit{history} \\ \hline
\end{array}$$

A move is illegal if it is to an occupied space,

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{GothelloIllegalMoveNonblank}} \\ \hline
\textit{GothelloAction} \\
\Xi \textit{GothelloPosition} \\ \hline
\textit{result!} = \textit{illegal\_move} \\
\exists\, s : \textit{SQUARE} \bullet \\
\qquad \textit{move?} = \textit{move}\ s \land \\
\qquad \textit{board}\ s \neq \textit{blank} \\ \hline
\end{array}$$

or if the stone placed becomes part of a group with no remaining liberties.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{GothelloIllegalMoveBlocked}} \\ \hline
\textit{GothelloAction} \\
\Xi \textit{GothelloPosition} \\ \hline
\textit{result!} = \textit{illegal\_move} \\
\exists\, s : \textit{SQUARE};\ \textit{cboard} : \textit{BOARD} \bullet \\
\qquad \textit{move?} = \textit{move}\ s \land \\
\qquad \textit{cboard} = \textit{board} \oplus \{ s \mapsto (\textit{stone}\ \textit{to\_move}) \} \land \\
\qquad \#(\textit{liberties}\ \textit{cboard}\ s) = 0 \\ \hline
\end{array}$$

Otherwise, a non-pass move will capture any opposing groups which have their last liberty removed, changing the color of these groups.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{GothelloMove}} \\
\hline
\textit{GothelloAction} \\
\textit{cboard} : \textit{BOARD} \\
\textit{capture\_at}, \textit{captures} : \textit{SQUARE} \rightarrow \textit{SQUARE} \pfun \textit{SQUAREVAL} \\
\hline
\textit{result}! = \textit{not\_done} \\
\forall\, s : \textit{SQUARE} \bullet \\
\quad \textit{capture\_at}\; s = \mathbf{if}\; \#(\textit{liberties cboard}\; s) = 0 \\
\quad\quad \mathbf{then}(\textit{group cboard}\; s) \times \{\textit{stone to\_move}\} \\
\quad\quad \mathbf{else}\, \varnothing \\
\forall\, s : \textit{SQUARE} \bullet \\
\quad \textit{captures}\; s = \bigcup(\textit{capture\_at} \llparenthesis (\textit{neighbor cboard}) \llparenthesis \{s\} \rrparenthesis \rrparenthesis) \\
\exists\, s : \textit{SQUARE} \bullet \\
\quad \textit{move}? = \textit{move}\; s \land \\
\quad \textit{board}\; s = \textit{blank} \land \\
\quad \textit{cboard} = \textit{board} \oplus \{s \mapsto (\textit{stone to\_move})\} \land \\
\quad \#(\textit{liberties cboard}\; s) > 0 \land \\
\quad \textit{board}' = \textit{cboard} \oplus (\textit{captures}\; s) \\
\textit{to\_move}' = \textit{opponent to\_move} \\
\hline
\end{array}
$$

For a pass, there are two possible outcomes. If the opponent did not also just pass, the game continues.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{GothelloPass}} \\
\hline
\textit{GothelloAction} \\
\Xi \textit{GothelloPosition} \\
\hline
\textit{result}! = \textit{not\_done} \\
\textit{move}? = \textit{pass} \\
\textit{history}\; 1 \neq \textit{pass} \\
\textit{to\_move}' = \textit{opponent to\_move} \\
\hline
\end{array}
$$

Otherwise, the game is over, with the result determined simply by which player has the most stones on the board.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{GothelloGameOver}} \\
\hline
\textit{GothelloAction} \\
\Xi \textit{GothelloPosition} \\
\textit{black\_stones}, \textit{white\_stones} : \mathbb{N} \\
\hline
\textit{result}! = \\
\quad \mathbf{if}\; \textit{black\_stones} > \textit{white\_stones}\; \mathbf{then}\; \textit{win black} \\
\quad \mathbf{else\,if}\; \textit{white\_stones} > \textit{black\_stones}\; \mathbf{then}\; \textit{win white} \\
\quad \mathbf{else}\; \textit{draw} \\
\textit{move}? = \textit{pass} \\
\textit{history}\; 1 = \textit{pass} \\
\textit{black\_stones} = \#(\textit{board} \rhd \{\textit{stone black}\}) \\
\textit{white\_stones} = \#(\textit{board} \rhd \{\textit{stone white}\}) \\
\hline
\end{array}
$$

A Gothello *turn* consists of any one of the five possible transitions

$$\begin{array}{l}
\mathit{GothelloTurn} \mathrel{\widehat{=}} \\
\qquad \mathit{GothelloIllegalMoveNonblank} \lor \\
\qquad \mathit{GothelloIllegalMoveBlocked} \lor \\
\qquad \mathit{GothelloMove} \lor \\
\qquad \mathit{GothelloPass} \lor \\
\qquad \mathit{GothelloGameOver}
\end{array}$$

The proof that these rules are well-founded remains to be completed. The initial state is well-defined. The five possible transitions in a Gothello turn are disjoint. It seems straightforward to show that one of the five transitions applies in every situation, and that the definition of each transition is well-founded and deterministic, which would essentially complete the proof.

Table 1: Greeting

name	response	meaning
greeting	000 Gothello ⟨*version-number*⟩	Greeting message

Table 2: Initial Requests

name	request	meaning
want_white	⟨*version*⟩ player white ⟨*optional-name*⟩	Will play white
want_black	⟨*version*⟩ player black ⟨*optional-name*⟩	Will play black
want_side	⟨*version*⟩ player ? ⟨*optional-name*⟩	Will play either
want_observe	⟨*version*⟩ observer ⟨*optional-name*⟩	Will observe

2 Server

The *Gothello* server listens on a port in the range 29068 . . . 29077 for a connection.[1] All input to the server will be in the form of ASCII text lines, terminated with a CR character (ASCII code 13). All server responses will be in the form of ASCII text lines, terminated with a CR and then an LF character (ASCII code 10). Responses will begin with a 3-digit numerical code, and be followed by whitespace and a (non-standard) explanatory text message. Requests and responses not currently implemented by the server will have their identifier in italics: those implemented will have boldface identifiers.

Any number of observers may connect to the server, as well as the two players. The server will always be in a state determined by the input it has seen. This state will determine which messages it will accept, and which responses it will return. The server may be in different states for different connections: it must synchronize the connections at key points.

$$
\begin{aligned}
&\mathit{STATE} ::= \mathit{initial} \mid \mathit{seated} \mid \mathit{playing} \mid \mathit{done}\\
&\mathit{ENTITY} ::= \mathit{player}\langle\!\langle \mathit{PLAYER} \rangle\!\rangle \mid \mathit{observer}\langle\!\langle \mathbb{N}_1 \rangle\!\rangle\\
&\mathit{observer} \in \operatorname{seq} \mathit{ENTITY}
\end{aligned}
$$

$$
\mid \mathit{cstate} : \mathit{ENTITY} \nrightarrow \mathit{STATE}
$$

Upon connection to the server, an entity will receive a greeting in the form indicated by Table 1. The version number is a pair of integers separated by a decimal point. This document describes version 0.9.

The initial message sent to the server must be as shown in Table 2. Responses are shown in Table 3. The ⟨*optional-name*⟩ is an optional double-quoted string (with the convention that two consecutive double-quotes ”” inside the string escape to a single double-quote ”) of up to 31 characters used to identify the entity. The version number is as above, and is used to identify the client version. The client version must be no greater (under the usual ordering) than the server version. If both players indicate “player ?”, the server will randomly select a white and black player.

Once both a white player and a black player have connected, the setup phase will be over. The server may indicate to each entity the other entities involved, by sending messages as shown in Table 4. ⟨*name*⟩ and ⟨*optional-name*⟩ are double-quoted strings as described below. ⟨*number*⟩ is a decimal number. (All entities should be prepared to deal with numbers up to 3 decimal digits, and to discard an arbitrary number).

The server will then signal the start of game by sending a message to each connected entity, as shown in Table 5.

After this, the server will accept moves from players in alternation, of the form shown in Table 6, where the ⟨*move number*⟩ is a standard decimal number indicating the ply of the move, the ⟨*ellipses-if-white*⟩ will be the string ... for a move by white and the empty string for a move by black, and ⟨*move*⟩ will be a move

[1] All numbers in this section will be base 10 (decimal) unless otherwise stated.

Table 3: Initial Responses

name	response	meaning
seat_granted	100	Request accepted
seat_granted_tc	101 ⟨*secs*⟩ ⟨*opp-secs*⟩	Request accepted with time controls
	19x	Request not accepted
seat_taken	191	Other player holds requested side
seat_full	192	There are already two players
seat_private	193	Cannot observe
seat_illegal	198	Illegal version number
seat_garbled	199	Request not understood

Table 4: Configuration Messages

name	response	meaning
config_white	341 ⟨*name*⟩	White player is ⟨*name*⟩
config_black	342 ⟨*name*⟩	Black player is ⟨*name*⟩
config_observer	343 ⟨*number*⟩ ⟨*name*⟩	Observer ⟨*number*⟩ is ⟨*name*⟩
config_nobserver	344 ⟨*number*⟩	There are ⟨*number*⟩ observers

in algebraic notation.

Instead of a move, the following inputs may also be accepted as shown in Table 7. Responses to actions are shown in Table 8.

After each accepted action, a message will be sent to each connected entity, as shown in Table 9. Upon termination of the game, the server will close all connections.

If the participant is an observer, every status message will be followed by two state display messages showing the current state of the game, as in Table 10. The times will be in seconds, and the ⟨*to-move*⟩ value will be either "b", "w", or "." indicating Black, White, or the game is over. The **sdisp_board** message will be immediately followed by 5 lines of 5 printable characters indicating the board state. Each character will be as above: "b", "w", or "." indicating a blank square.

Table 5: Starting Messages

name	response	meaning
role_white	351	You will play white
role_black	352	You will play black
role_observer	353	You will observe

Table 6: Move Syntax

name	**request**	**meaning**
action_move	⟨*move number*⟩ ⟨*ellipses-if-white*⟩ ⟨*move*⟩	Make a move

Table 7: Alternatives To Moving

name	**request**	**meaning**
action_resign	resign	Player resigns
action_pass	pass	Player passes

Table 8: Responses To Actions

name	**response**	**meaning**
	20x	Action accepted
result_continue	200	Continue playing
result_continue	207 ⟨*secs*⟩	Continue with time left
result_win	201	You win
result_lost	202	You lose
result_drawn	203	You draw
result_resigned	204	Resignation accepted
	29x	Action not accepted
result_illegal	291	Illegal request
result_garbled	299	Request not understood

Table 9: Status Messages

name	response	meaning
	31x	Game continues
status_moves_black	311 ⟨*move-number*⟩ ⟨*move*⟩	Black move
status_moves_white	312 ⟨*move-number*⟩ ... ⟨*move*⟩	White move
status_moves_black_tc	313 ⟨*move-number*⟩ ⟨*move*⟩ ⟨*secs*⟩	Black move and time
status_moves_white_tc	314 ⟨*move-number*⟩ ... ⟨*move*⟩ ⟨*secs*⟩	White move and time
status_passes_black	315 ⟨*move-number*⟩ pass	Black passes
status_passes_white	316 ⟨*move-number*⟩ ... pass	White passes
status_passes_black_tc	317 ⟨*move-number*⟩ pass ⟨*secs*⟩	Black pass and time
status_passes_white_tc	318 ⟨*move-number*⟩ ... pass ⟨*secs*⟩	White pass and time
	32x, 36x	Game over
status_winsmove_black	321 ⟨*move-number*⟩ ⟨*move*⟩	Black wins by move
status_losesmove_black	322 ⟨*move-number*⟩ ⟨*move*⟩	Black loses by move
status_winsmove_white	323 ⟨*move-number*⟩ ... ⟨*move*⟩	White wins by move
status_losesmove_white	324 ⟨*move-number*⟩ ... ⟨*move*⟩	White loses by move
status_drawsmove_black	325 ⟨*move-number*⟩ ⟨*move*⟩	Drawn by Black move
status_drawsmove_white	326 ⟨*move-number*⟩ ... ⟨*move*⟩	Drawn by White move
status_resigns_white	327	Black wins by resignation
status_resigns_black	328	White wins by resignation
status_flagfell_white	361	Black wins by White time expiring
status_flagfell_black	362	White wins by Black time expiring
	34x	(see above)
	35x	(see above)
	39x	Bad status
status_disconnect_black	391	Black disconnected
status_disconnect_white	392	White disconnected
status_garble	399	Unknown problem

Table 10: State Display Messages

name	response	meaning
	38x	state display
sdisp_status	380 ⟨*move-number*⟩ ⟨*to-move*⟩	state
sdisp_status_tc	381 ⟨*move-number*⟩ ⟨*time-b*⟩ ⟨*time-w*⟩ ⟨*to-move*⟩	state and time
sdisp_board	382	board