A Networked Awari Referee: Specification

Bart Massey

August 10, 2000

The many variants of the African game Awari are among the oldest known games of intellectual skill. In this document, we will describe a networked server to which human and computer players can connect to play the game in a refereed fashion.

1 The Game Of Awari

The rules of awari are complicated and have many variants. This discussion is based on the excellent summary of Schaeffer et al. [3].

Awari is a two player game, played by players conventionally designated as *north* and *south*.

$$PLAYER ::= north \mid south$$

$$\begin{array}{|l}
opponent : PLAYER \rightarrowtail\!\!\!\!\rightarrow PLAYER \\
\hline
opponent\ north = south \\
opponent\ south = north
\end{array}$$

The board, shown in figure 1, consists of 12 *pits* into which a number of *stones* are placed. In addition, two end pits, or *awari*, contain stones captured by the two sides during play. Starting from the NE corner and moving counterclockwise, the pits are named by the letters a through f, first in lowercase and then in uppercase. This convention is sensible, since the fundamental sowing operation described below moves counterclockwise.

$$PIT ::= a \mid b \mid c \mid d \mid e \mid f \mid A \mid B \mid C \mid D \mid E \mid F$$

$$\begin{array}{|l}
pit_order : \mathrm{seq}\, PIT \\
next : PIT \rightarrowtail\!\!\!\!\rightarrow PIT \\
prev : PIT \rightarrowtail\!\!\!\!\rightarrow PIT \\
\hline
pit_order = \langle a, b, c, d, e, f, A, B, C, D, E, F \rangle \\
\forall\, p : PIT \bullet next\, p = pit_order(((pit_order^{\sim})\, p) \bmod 12 + 1) \\
prev = next^{\sim}
\end{array}$$

The board starts with 48 stones, and stones are never lost from the game: all 48 must always be somewhere. It is thus useful to talk about the the total contents of a set of pits: more generically, about the total sum over the range of some relation.

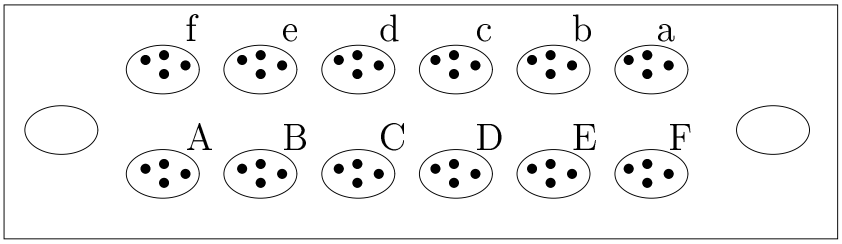

Figure 1: Awari Board In Initial Configuration

$[X]$

$\Sigma : (X \leftrightarrow \mathbb{N}) \rightarrow \mathbb{N}$

$\Sigma \varnothing = 0$

$\forall\, r : X \leftrightarrow \mathbb{N};\ x : X;\ n : \mathbb{N} \mid (x \mapsto n) \in r \bullet$
$\quad \Sigma r = n + \Sigma(r \setminus \{x \mapsto n\})$

The state of the board at any given time is essentially just the number of stones in each pit. The players alternate moves.

$BOARD == PIT \rightarrow \mathbb{N}$
$SCORE == PLAYER \rightarrow \mathbb{N}$

AwariBoard

$board : BOARD$
$score : SCORE$
$to\_move : PLAYER$

$\Sigma board + \Sigma score = 48$

Each pit initially contains 4 stones, as shown in figure 1. The *south* player moves first.

InitAwariBoard

$AwariBoard'$

$board' = PIT \times \{4\}$

$score' = PLAYER \times \{0\}$

$to\_move' = south$

The players alternate in *sowing* the stones in a pit of their choice on their side of the board. (*north* owns pits a–f, and *south* pits A–F.)

$side : PLAYER \rightarrowtail\!\!\!\!\rightarrow \mathbb{P}\, PIT$

$side\ north = \mathrm{ran}((1 \mathinner{..} 6) \lhd pit\_order)$
$side\ south = PIT \setminus (side\ north)$

A pit may be sown if it contains one or more stones: the stones are removed from the pit, and placed one at a time into subsequent pits, moving around the board in counterclockwise order. The original pit is skipped whenever it is encountered.

$$
\begin{array}{|l}
next\_skip : PIT \rightarrow PIT \rightarrow PIT \\
\hline
\forall p : PIT \bullet next\_skip\ p = next \oplus \{prev\ p \mapsto next\ p\}
\end{array}
$$

Thus, the sequence of pits to be sown is a prefix of the infinite cyclic sequence of pits which repeats every 11 elements after skipping the sown pit. We call this infinite sequence the *trace* of the sown pit.

$$
\begin{array}{|l}
trace : PIT \rightarrow \mathrm{seq}\, PIT \\
\hline
\forall p : PIT \bullet trace\ p = (\lambda\, n : \mathbb{N}_1 \bullet (iter\ n\, (next\_skip\ p))\ p)
\end{array}
$$

Thus we can sow the stones by emptying the pit to be sown and adding a stone to each other pit for each time it is hit by sowing. The last pit filled is the first pit we will examine for possible capture below.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Sow}} \\
\hline
\Delta AwariBoard \\
move : PIT \\
to\_capture : PIT \\
\hline
\forall p : PLAYER \bullet score\ p < 25 \\
move \in side\ to\_move \\
board\ move > 0 \\
to\_capture = (trace\ move)(board\ move) \\
board' = \\
\quad \textbf{let}\ emptied == board \oplus \{move \mapsto 0\}; \\
\quad sown == (1 \mathbin{..} (board\ move)) \lhd (trace\ move) \bullet \\
\qquad (\lambda\, p : PIT \bullet (emptied\ p) + \#(sown \rhd \{p\})) \\
score' = score \\
to\_move' = to\_move \\
\hline
\end{array}
$$

When sowing a pit, if the last pebble placed makes a group of two or three, then that pit's stones are *captured* and scored by placing in the capturing player's awari. If the previous pit then contains a group of two or three stones, these stones are also captured, and so forth. Thus, the set of pits which are captured is the set of pits on the opponent's side of the board reachable by captures from the last pit sown: This set is given by by the range of the transitive closure of the next relation restricted to the capturable elements.

$$
\begin{array}{|l}
\multicolumn{1}{l}{[X]} \\
\hline
reachable : (X \leftrightarrow X) \rightarrow \mathbb{P}\, X \rightarrow \mathbb{P}\, X \\
\hline
\forall r : X \leftrightarrow X;\ xs : \mathbb{P}\, X \bullet \\
\quad reachable\ r\ xs = \mathsf{ran}(xs \lhd (r^*)) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{Capture} \\
\Delta AwariBoard \\
to\_capture : PIT \\
captures : \mathbb{P}\, PIT \\
\hline
captures = \\
\quad \mathbf{let}\ vul == \mathsf{dom}(board \rhd \{2,3\}) \cap side(opponent\ to\_move) \bullet \\
\qquad reachable\ (vul \lhd prev \rhd vul)\ \{to\_capture\} \\
board' = board \oplus (captures \times \{0\}) \\
score'\ to\_move = score\ to\_move + \Sigma(captures \lhd board) \\
score'(opponent\ to\_move) = score(opponent\ to\_move) \\
to\_move' = to\_move \\
\hline
\end{array}
$$

The sowing of stones to capture all stones on the opponent's side of the board is known as a *grand slam*, clean sweep, or grand coup. Normally, a grand slam ends the game, capturing all stones remaining on the board. The rules governing the grand slam vary widely, however: Schaeffer *et. al.* [3] lists a number of variations:

a) Stones may not be sown for a grand slam (unless no other move is possible).

b) The stones may be sown for a grand slam, but no capture results.

c) The stones may be sown for a grand slam, but the last pit is not captured.

d) The stones may be sown for a grand slam, but only the first pit is captured.

e) The stones may be sown for a grand slam, and all captures happen: the remaining stones on the board are awarded to the opponent.

This paper will consider only variant (e), which will be used in the August 2000 Mind Sports Olympiad [2] computer competition. The 1990–1992 Computer Olympiads [1] used variation (a). Many human players prefer the simpler version.

There are a number of possible outcomes of a (attempt to) move.

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

$$
\begin{array}{|l}
winner : SCORE \rightarrow RESULT \\
\hline
\forall\, s : SCORE \bullet winner\ s = \\
\quad \mathbf{if}\ s\ north > s\ south \\
\qquad \mathbf{then}\ win\ north \\
\quad \mathbf{else\ if}\ s\ south > s\ north \\
\qquad \mathbf{then}\ win\ south \\
\quad \mathbf{else}\ draw
\end{array}
$$

A game may be ended by a player being unable to move, in which case the remaining stones on the board belong to the opponent.

$$
\begin{array}{|l}
\hline
\textit{ClaimStones} \\
score, score' : SCORE \\
board : BOARD \\
\hline
\forall\, p : PLAYER \bullet score'\ p = score\ p + \Sigma(side\ p \lhd board) \\
\hline
\end{array}
$$

However, a player must leave the opponent with a legal move at the start of their turn, if it is possible to do so.

A game will also be ended by repetition of position. (This is generally only true of computer play. In human play, it is more common to end by agreement of both sides.) The most common rule here is that each player captures the stones on their side of the board. One popular variant, used in the 1990–1992 Computer Olympiad, is to not count the stones left on the board at the end. To keep track of repetitions, it is necessary to keep track of the set of positions seen so far.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Positions}} \\
\hline
\textit{positions} : \mathbb{P}\, \textit{AwariBoard} \\
\hline
\end{array}$$

To summarize the variant of interest here: a move in the game consists of altering the board and recording the resulting board position. Moves alternate between players.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AwariMove}} \\
\hline
\Delta \textit{AwariBoard} \\
\Delta \textit{Positions} \\
\textit{opp_stones} : \textit{BOARD} \rightarrow \mathbb{N} \\
\textit{result} : \textit{RESULT} \\
\hline
\textit{positions}' = \textit{positions} \cup \{\theta \textit{AwariBoard}\} \\
\forall\, b : \textit{BOARD} \bullet \textit{opp_stones}\ b = \Sigma((\textit{side}(\textit{opponent}\ \textit{to_move})) \lhd b) \\
\hline
\end{array}$$

If either player has 25 or more stones at the start of their turn, the player with the most stones wins, and the game is over.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{WinMargin}} \\
\hline
\textit{AwariMove} \\
\hline
\exists\, p : \textit{PLAYER} \bullet \textit{score}\ p \geq 25 \\
\textit{score}' = \textit{score} \\
\textit{result} = \textit{winner}\ \textit{score}' \\
\hline
\end{array}$$

If the sowing is impossible, or the position is a repetition of a previous one, each player claims the stones on their side of the board, and the game is over.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{NoMove}} \\
\hline
\textit{AwariMove} \\
\textit{ClaimStones} \\
\hline
\forall\, p : \textit{PLAYER} \bullet \textit{score}\ p < 25 \\
\theta \textit{AwariBoard} \in \textit{positions}\ \vee \\
\Sigma(\textit{side}\ \textit{to_move} \lhd \textit{board}) = 0 \\
\textit{result} = \textit{winner}\ \textit{score}' \\
\hline
\end{array}$$

If the opponent's pits are all empty, the player must sow a pit which leaves the opponent a move if possible.

$$
\begin{array}{|l}
\hline
\textit{NormalMove} \\
\textit{AwariMove} \\
\textit{Sow} \mathbin{\S} \textit{Capture} \\
\hline
\forall b : \textit{BOARD} \bullet \textit{opp\_stones board} = 0 \Rightarrow \textit{opp\_stones board}' > 0 \\
\textit{result} = \textit{not\_done} \\
\hline
\end{array}
$$

A grand slam capture forfeits all stones still on the board at the end of the turn.

$$
\begin{array}{|l}
\hline
\textit{ForfeitStones} \\
\textit{score}, \textit{score}' : \textit{SCORE} \\
\textit{board} : \textit{BOARD} \\
\hline
\forall p : \textit{PLAYER} \bullet \textit{score}'\, p = \textit{score}\, p + \Sigma(\textit{side}(\textit{opponent}\, p) \lhd \textit{board}) \\
\hline
\end{array}
$$

A grand slam is the capture the rest of the opponent's stones.

$$
\begin{array}{|l}
\hline
\textit{GrandSlam} \\
\textit{AwariMove} \\
\textit{Sow} \mathbin{\S} \textit{Capture} \mathbin{\S} \textit{ForfeitStones} \\
\hline
\textit{opp\_stones board} > 0 \\
\textit{opp\_stones board}' = 0 \\
\textit{result} = \textit{winner score}' \\
\hline
\end{array}
$$

A move can only be illegal if a move is available and neither side has won. An illegal move is from a pit on the wrong side of the board, a pit with no stones, or fails to feed an opponent with stones when necessary. (Note that this schema does not assume *NormalMove* is deterministic for a given move, although we know it to be.)

$$
\begin{array}{|l}
\hline
\textit{IllegalMove} \\
\textit{AwariMove} \\
\hline
\forall p : \textit{PLAYER} \bullet \textit{score}\, p < 25 \\
\exists p : \textit{side to\_move} \bullet \textit{board}\, p > 0 \\
\quad \textit{move} \notin \textit{side to\_move} \lor \textit{board move} = 0 \lor \\
\quad \textit{opp\_stones board} = 0 \land \\
\qquad (\forall m : \textit{NormalMove} \mid m.\textit{move} = \textit{move} \bullet \\
\qquad\quad \textit{opp\_stones}\ m.\textit{board}' = 0) \land \\
\qquad (\exists m : \textit{NormalMove} \mid m.\textit{move} \neq \textit{move} \bullet \\
\qquad\quad \textit{opp\_stones}\ m.\textit{board}' > 0) \\
\textit{result} = \textit{illegal\_move} \\
\hline
\end{array}
$$

The referee reports whether a move continues the game, ends the game, or is illegal.

$$
\begin{array}{l}
\textit{AwariRef} \mathrel{\widehat{=}} \textit{NormalMove} \lor \textit{WinMargin} \lor \\
\quad \textit{NoMove} \lor \textit{GrandSlam} \lor \textit{IllegalMove}
\end{array}
$$

The game starts with the initial board, and no positions yet seen.

Table 1: Greeting

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

Table 2: Initial Requests

name	request	meaning
want_north	⟨*version*⟩ player north ⟨*optional-name*⟩	Will play north
want_south	⟨*version*⟩ player south ⟨*optional-name*⟩	Will play south
want_side	⟨*version*⟩ player ? ⟨*optional-name*⟩	Will play either
want_observe	⟨*version*⟩ observer ⟨*optional-name*⟩	Will observe

InitAwari

$InitAwariBoard$
$Positions'$

$positions' = \varnothing$

2 Server

The Awari server listens on a port in the range 29046 . . . 29056 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.

$$STATE ::= initial \mid seated \mid playing \mid done$$
$$ENTITY ::= player\langle\langle PLAYER\rangle\rangle \mid observer\langle\langle \mathbb{N}_1 \rangle\rangle$$
$$observer \in \mathbb{P}(\mathrm{seq}\, ENTITY)$$

$$cstate : ENTITY \rightarrow\!\!\!\!\!\!+\; STATE$$

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 north and south player.

Once both a north player and a south 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

[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_north	341 ⟨*name*⟩	North player is ⟨*name*⟩
config_south	342 ⟨*name*⟩	South player is ⟨*name*⟩
config_observer	343 ⟨*number*⟩ ⟨*name*⟩	Observer ⟨*number*⟩ is ⟨*name*⟩
config_nobserver	344 ⟨*number*⟩	There are ⟨*number*⟩ observers

⟨*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-north*⟩ will be the string ... for a move by north and the empty string for a move by south, and ⟨*move*⟩ will be a move in the notation described above.

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.

The draw protocol is as one would expect: a draw may be offered by a player whose turn it is to move. If it is rejected by the other player, it may not be offered again until the next move.

References

[1] D. Levy and D. Beal, editors. *Heuristic Programming in Artificial Intelligence 3: The Third Computer Olympiad.* Ellis-Horwood, 1992.

[2] Mind Sports Olympiad. Web document, 2000. URL `http://www.msoworld.com/mindsports.html`.

[3] Jonathan Schaeffer et al. The Rules Of Awari. Web document, 1998. URL `http://www.cs.ualberta.ca/awari/rules.html`.

Table 5: Starting Messages

name	response	meaning
role_north	351	You will play north
role_south	352	You will play south
role_observer	353	You will observe

Table 6: Move Syntax

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

Table 7: Alternatives To Moving

name	request	meaning
action_resign	resign	Player resigns
action_draw_req	draw?	Player offers a draw
action_draw_ack	draw	Player accepts a draw
action_draw_nak	nodraw	Player refuses a draw

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
result_drawreq	205	Received draw request
result_drawnak	206	Received draw refusal
	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_south	311 ⟨*move-number*⟩ ⟨*move*⟩	South move
status_moves_north	312 ⟨*move-number*⟩ ... ⟨*move*⟩	North move
status_moves_south_tc	313 ⟨*move-number*⟩ ⟨*move*⟩ ⟨*secs*⟩	South move and time
status_moves_north_tc	314 ⟨*move-number*⟩ ... ⟨*move*⟩ ⟨*secs*⟩	North move and time
	32x, 36x	Game over
status_winsmove_south	321 ⟨*move-number*⟩ ⟨*move*⟩	South wins by move
status_losesmove_south	322 ⟨*move-number*⟩ ⟨*move*⟩	South loses by move
status_winsmove_north	323 ⟨*move-number*⟩ ... ⟨*move*⟩	North wins by move
status_losesmove_north	324 ⟨*move-number*⟩ ... ⟨*move*⟩	North loses by move
status_drawsmove_south	325 ⟨*move-number*⟩ ⟨*move*⟩	Drawn by South move
status_drawsmove_north	326 ⟨*move-number*⟩ ... ⟨*move*⟩	Drawn by North move
status_resigns_north	327	South wins by resignation
status_resigns_south	328	North wins by resignation
status_draw_agreed	329	Draw accepted
status_flagfell_north	361	South wins by North time expiring
status_flagfell_south	362	North wins by South time expiring
	33x	Draw protocol
status_reqsdraw_south	331	Draw requested by South
status_reqsdraw_north	332	Draw requested by North
status_refsdraw_south	333	Draw refused by South
status_refsdraw_north	334	Draw refused by North
	34x	(see above)
	35x	(see above)
	39x	Bad status
status_disconnect_south	391	South disconnected
status_disconnect_north	392	North disconnected
status_garble	399	Unknown problem