Specification of the Quoridor Text Protocol

December, 2015

http://quoridor.di.uoa.gr/

Avlonitis Spyros && Giannos Chatziagapis

Back to top

Contents

Protocol Basics

1.1 Character Set

All messages exchanged in this protocol are to be considered as 8-bit character sequences. Only characters in the US-ASCII character set (ANSI X3.4-1986) are used for standardized commands and responses. Other characters may be used in comments (section 1.9) but there is no preferred character set specified for those.

1.2 Control Characters

Character values 0-31 and 127 are control characters in ASCII. The following control characters have a specific meaning in the protocol:

HT (dec 9) Horizontal Tab CR (dec 13) Carriage Return LF (dec 10) Line Feed

All other control characters must be discarded on input and should not be used on output.

1.3 Whitespace

The following ASCII characters can be used to indicate whitespace in the protocol:

SPACE (dec 32) Space HT (dec 9) Horizontal Tab

In the rest of the specification we use 'space' to denote a whitespace character. On input this may be either a SPACE or a HT. On output only a SPACE should be used.

1.4 Newline Convention

A newline is indicated by a single LF character. Any occurence of a CR character must be discarded on input, both by the engine and the controller. On output either LF or some combination of CR and LF can be used. In syntax descriptions we use \n to indicate a newline.

1.5 Command Structure

A command is exactly one line long, with the syntax

```
command_name [arguments]
```

1.6 Response Structure

If successful, the engine returns a response of the form

```
= result
```

Here '=' indicates success and result is a piece of text ending with two consecutive newlines.

1.7 Error Messages

If unsuccessful, the engine returns a response of the form

```
? error_message
```

Here '?' indicates failure and error_message gives an explanation for the failure, also ending with two consecutive newlines.

1.8 Timing

There are no synchronization requirements between the controller and the engine. The controller may send commands at any time, regardless of whether it has obtained responses for previous commands. The engine may send responses whenever they are ready. It must, however, respond to the commands in the same order as they come in. The engine is allowed to make pauses while sending a response.

1.9 Comments

Comments can be included in the command stream. All text between a hash sign (#) and the following newline is considered as comments and should be discarded on input.

1.10 Empty lines

Empty lines and lines with only whitespace sent by the controller must be ignored by the engine. No response must be generated. Empty lines and lines with only whitespace sent by the engine and occuring outside a response must be ignored by the controller. Notice that pure comment lines will appear as empty lines after the comment has been discarded.

1.11 Board Coordinates

Board intersections, in this document called vertices, are encoded by a letter plus a number. On a 9x9 board the letters go from A to I from the left to the right. The numbers go from 9 to 1, from the top to the bottom. Thus the upper left corner is called A9, the upper right corner I9, the lower left corner A1, and the lower right corner I1. Smaller boards use the obvious subset of these coordinates. Boards larger than 25x25 are not supported by the protocol.

1.12 Panic Situations

If an engine for some reason, e.g. an internal error, finds itself in a position where it cannot meaningfully continue the session, the correct action is to just close the connection. This is also what typically will happen if the program should happen to encounter an uncontrolled crash.

Protocol Details

2.1 Preprocessing

When a command string arrives to an engine, it is expected to perform the following four operations before any further parsing takes place:

Remove all occurences of CR and other control characters except for HT and LF.

For each line with a hash sign (#), remove all text following and including this character.

Convert all occurences of HT to SPACE.

Discard any empty or white-space only lines.

When a response arrives to a controller, it is expected only to do steps 1 and 3 above.

Naturally an implementation does not have to actually do this preprocessing as a separate step but may interleave it with other parts of the parsing. For purposes of the following specifications, though, the preprocessing is supposed to have been carried out in full.

2.2 Simple Entities

- **int**

  An int is an unsigned integer in the interval $0 <= x <= 2^{31} - 1.

- **string**

  A string is a sequence of printable, non-whitespace characters. Strings are case sensitive.

- **vertex**

  A vertex is a board coordinate consisting of one letter and one number, as defined in section 1.11. Vertices are not case sensitive. Examples: ``B13", ``j11".

- **color**

  A color is one of the strings ``white" or ``w" to denote white, or ``black" or ``b" to denote black. Colors are not case sensitive.

- **orientation**

  A orientation is one of the strings ``horizontal" or ``h" to denote horizontal, or ``vertical" or ``v" to denote vertical . Orientations are not case sensitive.

- **move**

  A move is the combination of one color and one vertex, separated by space. Moves are not case sensitive. Examples: ``white h3", ``B F5".

- **wall**

  A wall is the combination of one color , one vertex (northwest of the wall center) and one orientation, separated by space. Placements are not case sensitive. Examples: ``white h3 h", ``B F5 VERTICAL".

- **boolean**

  A boolean is one of the strings ``false" or ``f" to denote false and ``true" or ``t" to denote true. Booleans are not case sensitive.

2.2.1 Compound Entities

- **List**

  An x* is a space separated list of entities of type x, where x may be any of the entities specified so far. The list can have an arbitrary number of elements and goes on until an LF is encountered.

- **Multiline list**

  An x& is an LF separated list of entities of type x, where x may be any of the entities specified so far. The multiline list can have an arbitrary number of lines and goes on until two consecutive LFs are encountered.

2.3 Commands

A command has one of the syntaxes

```
command_name arguments\n
command_name\n
```

Command_name is a string. arguments is a space separated list of some entities, the composition of which varies with the command If arguments is missing it counts as empty.

2.4 Success Responses

A successful response has one of the syntaxes

```
=response\n\n
=\n\n
```

Response is a list of some entities, separated by space or a single LF, the composition of which varies with the command. The response may be empty.

2.5 Failure Responses

An unsuccessful response has the syntax

```
? error_message\n\n
```

Error_message is a string.

2.6 Standard Error Messages

If the engine receives an unknown or unimplemented command, use the error message ``unknown command". Some commands fail in certain cases with standardized error messages. Those are listed in the command descriptions in section 4. For other failures the engine can freely choose error message.

Internal State

3.1 State Variables

An engine is expected to keep track of the following state information:

```
board size
board configuration
number of walls of either color
game history
```

3.2 Default State

There is no default state for any state variable. When first started, the engine may set these as it likes. A controller which has some specific opinion about these values must set them explicitly with the appropriate commands

3.3 State Maintenance

The state is changed by certain commands, as specified in their description in section 4. State which is not explicitly modified must remain unchanged. A failed command must never change any state.

List of All Commands

4.1 Adminstrative Commands

```
name
arguments   none
effects     none
output      name
    string* name - Name of the engine
fails       never
comments    E.g. ``IP Quoridor''.


known_command
arguments   command_name
```

```
        string command_name - Name of a command
    effects     none
    output      known
        boolean known - ``true'' if the command is known by the engine, ``false''
otherwise
    fails       never
    comments    The protocol makes no distinction between unknown commands and known
but unimplemented ones.
                Do not declare a command as known if it is known not to work.
```

list_commands

```
    arguments   none
    effects     none
    output      commands
        string& commands - List of commands, one per row
    fails       never
    comments    Include all known commands, including required ones and private
extensions.
```

quit

```
    arguments   none
    effects     The session is terminated and the connection is closed.
    output      none
    fails       never
    comments    The full response of this command must be sent before the engine closes
the connection.
                The controller must receive the response before the connection is
closed on its side.
```

4.2 Setup Commands

boardsize

```
    arguments   size
        int size - New size of the board.
    effects     The board size is changed.
                The board configuration, the number of walls of each player, and move
history become arbitrary.
    output      none
    fails       Syntax error. If the engine cannot handle the new size, fails with the
error message "unacceptable size".
    comments    The controller must call clear_board and walls explicitly.
                Even if the new board size is the same as the old one, the board
configuration and the walls number become arbitrary.
```

clear_board

```
    arguments   none
    effects     The board is cleared, the two pawns return to their starting position,
                the number of walls of each player become arbitrary and the move
history is reset to empty.
    output      none
    fails       never
    comments
```

walls

```
    arguments   number_of_walls
                int number_of_walls - Number of walls for each player.
    effects     Each player has that number of walls at the begining
    output      none
    fails       never
    comments
```

4.3 Core Play Commands

```
    playmove
    arguments   move
        move move - Color and vertex of the move
    effects     The player of the requested color is played at the requested vertex.
                The number of captured stones is updated if needed and the move is
added to the move history.
    output      none
    fails       syntax error, illegal move. In the latter case, fails with the error
message ``illegal move''.
    comments    Consecutive moves of the same color are not considered illegal from the
protocol point of view.


    playwall
    arguments
        wall placement  - Color , vertex and orientation of the wall
    effects     A wall place at the requested vertex and orientation . Decrease number
of wall for that player by one
    output      none
    fails       syntax error, illegal placement. In the latter case, fails with the
error message ``illegal move''.
    comments    Consecutive moves of the same color are not considered illegal from the
protocol point of view.


    genmove
    arguments   color
        color color - Color for which to generate a move.
    effects     The engine makes a move or wall placement for the requested color.
    output      vertex orientation
        vertex||string  - Vertex where the move was played or where the wall was
placed.
    fails       never
    comments
    If orientation returned a wall placed, otherwise it's an move.


    undo
    arguments   times(optional)
            int times - how many times to undo
    effects     The game goes 'times' moves back.
    output      none
    fails       If times is greater than moves played or no moves are played. Fails
with the error message ``cannot undo''
    comments
```

4.4 Tournament Commands

```
    winner
    arguments   none
    effects     none
    output boolean color
        boolean - true if game ended , otherwise false
        color - It's the color of the winner
    fails       never
    comments
```

4.5 Debug Commands

```
    showboard
    arguments   none
    effects     none
    output      board
        string*& board - A diagram of the board position.
    fails       never
    comments    The engine may draw the board as it likes.
                It is, however, required to place the coordinates as described in
section 1.11.
                This command is only intended to help humans with debugging and the
output should never
                need to be parsed by another program.
```

Last Update 14/1/2016