Package ‘stockfish’

October 14, 2022

Title Analyze Chess Games with the 'Stockfish' Engine

Version 1.0.0

Description An implementation of the UCI open communication protocol that ships with 'Stockfish' <https://stockfishchess.org/>, a very popular, open source, powerful chess engine written in C++.

License GPL (>= 3)

URL https://github.com/curso-r/stockfish

BugReports https://github.com/curso-r/stockfish/issues

Imports processx, R6

Suggests covr, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.1.2

SystemRequirements C++17

NeedsCompilation yes

Author Caio Lente [aut, cre] (<https://orcid.org/0000-0001-8473-069X>),
Tord Romstad [ctb],
Marco Costalba [ctb],
Joona Kiiski [ctb],
Gary Linscott [ctb]

Maintainer Caio Lente <clente@curso-r.com>

Repository CRAN

Date/Publication 2022-03-19 00:00:02 UTC

R topics documented:

`fish` *Stockfish engine*

Description

This class represents a Stockfish engine, allowing the user to send commands and receive outputs according to the UCI protocol. In short, a `fish` object, when created, spawns a detached Stockfish process and pipes into its stdin and stdout.

Bundled Stockfish

This package comes bundled with Stockfish, a very popular, open source, powerful chess engine written in C++. It can achieve an ELO of 3544, runs on Windows, macOS, Linux, iOS and Android, and can be compiled in less than a minute.

When installing `{stockfish}` (lower case), Stockfish's (upper case) source code is compiled and the resulting executable is stored with your R packages. This is not a system-wide installation! You don't have to give it administrative privileges to run or even worry about any additional software.

But there are two main downsides:

- While the bundled version of the engine (Stockfish 14.1) is up-to-date as of March 2022, it isn't able to update itself. This means that, if I'm not able to port an upcomming version of Stockfish, the package will stay behind. Luckly, you can always download the version of your choosing and pass the executable as an argument to `new()`.
- Since version 12, Stockfish supports NNUE evaluation but this requires some pretty heavy binaries. In order to avoid bundling large files with `{stockfish}`, **I have decided to disable NNUE** in the source code. Again, you are free to download a NNUE-capable version and use it instead of the bundled executable.

UCI Protocol

UCI (Universal Chess Interface) is an open communication protocol that enables chess engines to communicate with user interfaces. Strictly speaking, this class implements the UCI protocol as publicized by Stefan-Meyer Kahlen, just with a focus on the Stockfish engine. This means that some methods are not implemented (see Common Gotchas) and that all tests are run on Stockfish, but everything else should work fine with other engines.

The quoted text at the end of the documentation of each method was extracted directly from the official UCI protocol, so you can see exactly what that command can do. In general, the commands are pretty self-explanatory, except for long algebraic notation (LAN), the move notation used by UCI. In this notation, moves are recorded using the starting and ending positions of each piece, e.g. e2e4, e7e5, e1g1 (white short castling), e7e8q (for promotion), 0000 (nullmove).

Implementation

All the heavy lifting of this class is done by the `{processx}` package. The Stockfish process is created with `processx::process$new` and IO is done with `write_input()` and `read_output()`. An important aspect of the communication protocol of any UCI engine is waiting for replies, and here this is done with a loop that queries the process with `poll_io()` and stops once the output comes back empty.

Before implementing the UCI protocol manually, this package used `{bigchess}`. It is a great package created by @rosawojciech, but it has some dependencies that are beyond the scope of this package and ultimately I wanted more control over the API (e.g. using `{R6}`).

Common Gotchas

This class has some specifics that the user should keep in mind when trying to communicate with Stockfish. Some of them are due to implementation choices, but others are related to the UCI protocol itself. This is by no means a comprehensive list (and you should probably read UCI's documentation), but here are a few things to look out for:

- Not every UCI method is implemented: since `{stockfish}` was made with Stockfish in mind, a couple of UCI methods that don't work with the engine were not implemented. They are `debug()` and `register()`.
- Most methods return silently: since most UCI commands don't output anything or output boilerplate text, most methods return silently. The exceptions are `run()`, `isready()`, `go()` and `stop()`; you can see exactly what they return by reading their documentations.
- Not every Stockfish option will work: at least when using the bundled version of Stockfish, not every documented option will work with `setoption()`. This happens because, as described above, the bundled version has some limitations. Options that will not work are labeled with an asterisk.
- Times are in milliseconds: unlike most R functions, every method that takes a time interval expects them in milliseconds, not seconds.

Public fields

`process` Connection to `{processx}` process running the engine

`output` String vector with the output of the last command

`log` String vector with the all outputs from the engine

Methods

Public methods:

- `fish$new()`
- `fish$run()`
- `fish$uci()`
- `fish$isready()`
- `fish$setoption()`
- `fish$ucinewgame()`
- `fish$position()`
- `fish$go()`
- `fish$stop()`
- `fish$ponderhit()`
- `fish$quit()`

- `fish$print()`
- `fish$clone()`

Method **`new()`:** Start Stockfish engine

By default, this function uses the included version of Stockfish. If you want to run a more recent version, you can pass its executable as an argument. For more information, see the Bundled Stockfish section of this documentation.

Usage:

```
fish$new(path = NULL)
```

Arguments:

`path` Path to Stockfish executable (defaults to bundled version)

Method **`run()`:** Send a command to be run on the engine.

Every supported command is documented on the UCI protocol as publicized by Stefan-Meyer Kahlen. Please refrain from sending more than one command per call as the engine can get confused! Also note that commands automatically get a newline (`\n`) at the end, so there is no need to append that to the string itself.

Usage:

```
fish$run(command, infinite = FALSE)
```

Arguments:

`command` A string with the command to run

`infinite` Whether the command involves `go infinite` (will make function return instantly as output should only be collected when a `stop()` command is issued)

Returns: A string vector with the output of the command or `NULL`

Method **`uci()`:** Tell the engine to use the UCI.

"Tell engine to use the uci (universal chess interface), this will be send once as a first command after program boot to tell the engine to switch to uci mode. After receiving the uci command the engine must identify itself with the 'id' command and sent the 'option' commands to tell the GUI which engine settings the engine supports if any. After that the engine should sent 'uciok' to acknowledge the uci mode. If no uciok is sent within a certain time period, the engine task will be killed by the GUI."

Usage:

```
fish$uci()
```

Method **`isready()`:** Ask if the engine is ready for more commands.

"This is used to synchronize the engine with the GUI. When the GUI has sent a command or multiple commands that can take some time to complete, this command can be used to wait for the engine to be ready again or to ping the engine to find out if it is still alive. E.g. this should be sent after setting the path to the tablebases as this can take some time. This command is also required once before the engine is asked to do any search to wait for the engine to finish initializing. This command must always be answered with 'readyok' and can be sent also when the engine is calculating in which case the engine should also immediately answer with 'readyok' without stopping the search."

Usage:

`fish$isready()`

Returns: Boolean indicating whether the output is exactly `"readyok"`

Method `setoption()`**:** Change the internal parameters of the engine. All currently supported options (according to Stockfish's documentation) are listed below, but note that **those marked with an * will not work** with the buldled version:

- `Threads`: The number of CPU threads used for searching a position. For best performance, set this equal to the number of CPU cores available.
- `Hash`: The size of the hash table in MB. It is recommended to set Hash after setting Threads.
- `Clear Hash`: Clear the hash table.
- `Ponder`: Let Stockfish ponder its next move while the opponent is thinking.
- `MultiPV`: Output the N best lines (principal variations, PVs) when searching. Leave at 1 for best performance.
- `Use NNUE*`: Toggle between the NNUE and classical evaluation functions. If set to "true", the network parameters must be available to load from file (see also EvalFile), if they are not embedded in the binary.
- `EvalFile*`: The name of the file of the NNUE evaluation parameters. Depending on the GUI the filename might have to include the full path to the folder/directory that contains the file. Other locations, such as the directory that contains the binary and the working directory, are also searched.
- `UCI_AnalyseMode`: An option handled by your GUI.
- `UCI_Chess960`: An option handled by your GUI. If true, Stockfish will play Chess960.
- `UCI_ShowWDL`: If enabled, show approximate WDL statistics as part of the engine output. These WDL numbers model expected game outcomes for a given evaluation and game ply for engine self-play at fishtest LTC conditions (60+0.6s per game).
- `UCI_LimitStrength`: Enable weaker play aiming for an Elo rating as set by UCI_Elo. This option overrides Skill Level.
- `UCI_Elo`: If enabled by UCI_LimitStrength, aim for an engine strength of the given Elo. This Elo rating has been calibrated at a time control of 60s+0.6s and anchored to CCRL 40/4.
- `Skill Level`: Lower the Skill Level in order to make Stockfish play weaker (see also UCI_LimitStrength). Internally, MultiPV is enabled, and with a certain probability depending on the Skill Level a weaker move will be played.
- `SyzygyPath`: Path to the folders/directories storing the Syzygy tablebase files. Multiple directories are to be separated by ";" on Windows and by ":" on Unix-based operating systems. Do not use spaces around the ";" or ":". Example: `C:\tablebases\wdl345;C:\tablebases\wdl6;D:\tablebases\c` #nolint It is recommended to store .rtbw files on an SSD. There is no loss in storing the .rtbz files on a regular HDD. It is recommended to verify all md5 checksums of the downloaded tablebase files (`md5sum -c checksum.md5`) as corruption will lead to engine crashes.
- `SyzygyProbeDepth`: Minimum remaining search depth for which a position is probed. Set this option to a higher value to probe less aggressively if you experience too much slowdown (in terms of nps) due to tablebase probing.
- `Syzygy50MoveRule`: Disable to let fifty-move rule draws detected by Syzygy tablebase probes count as wins or losses. This is useful for ICCF correspondence games.
- `SyzygyProbeLimit`: Limit Syzygy tablebase probing to positions with at most this many pieces left (including kings and pawns).

- `Move Overhead`: Assume a time delay of x ms due to network and GUI overheads. This is useful to avoid losses on time in those cases.
- `Slow Mover`: Lower values will make Stockfish take less time in games, higher values will make it think longer.
- `nodestime`: Tells the engine to use nodes searched instead of wall time to account for elapsed time. Useful for engine testing.
- `Debug Log File`: Write all communication to and from the engine into a text file.

"This is sent to the engine when the user wants to change the internal parameters of the engine. For the 'button' type no value is needed. One string will be sent for each parameter and this will only be sent when the engine is waiting. The name of the option in should not be case sensitive and can includes spaces like also the value. The substrings 'value' and 'name' should be avoided in and to allow unambiguous parsing, for example do not use = 'draw value'. Here are some strings for the example below:

- `setoption name Nullmove value true\n`
- `setoption name Selectivity value 3\n`
- `setoption name Style value Risky\n`
- `setoption name Clear Hash\n`
- `setoption name NalimovPath value c:\chess\tb\4;c:\chess\tb\5\n"`

Usage:

```
fish$setoption(name, value = NULL)
```

Arguments:

`name` Name of the option

`value` Value to set (or `NULL` if option doesn't need a value)

Method `ucinewgame()`**:** Tell the engine that the next search will be from a different game.

"This is sent to the engine when the next search (started with 'position' and 'go') will be from a different game. This can be a new game the engine should play or a new game it should analyse but also the next position from a testsuite with positions only. If the GUI hasn't sent a 'ucinewgame' before the first 'position' command, the engine shouldn't expect any further ucinewgame commands as the GUI is probably not supporting the ucinewgame command. So the engine should not rely on this command even though all new GUIs should support it. As the engine's reaction to 'ucinewgame' can take some time the GUI should always send 'isready' after ucinewgame to wait for the engine to finish its operation."

Usage:

```
fish$ucinewgame()
```

Method `position()`**:** Set up the position on the internal board. When passing a sequence of moves, use long algebraic notation (LAN) as described in the UCI Protocol section of this documentation.

"Set up the position described in fenstring on the internal board and play the moves on the internal chess board. if the game was played from the start position the string 'startpos' will be sent. Note: no 'new' command is needed. However, if this position is from a different game than the last position sent to the engine, the GUI should have sent a 'ucinewgame' inbetween."

Usage:

```
fish$position(position = NULL, type = c("fen", "startpos"))
```

Arguments:

`position` String with position (either a FEN or a sequence of moves separated by spaces)

`type` Either `"fen"` or `"startpos"`, respectively

Method `go()`**:** Start calculating on the current position.

"Start calculating on the current position set up with the 'position' command. There are a number of commands that can follow this command, all will be sent in the same string. If one command is not send its value should be interpreted as it would not influence the search.

- `searchmoves`: restrict search to this moves only. Example: after 'position startpos' and 'go infinite searchmoves e2e4 d2d4' the engine should only search the two moves e2e4 and d2d4 in the initial position.
- `ponder`: start searching in pondering mode. Do not exit the search in ponder mode, even if it's mate! This means that the last move sent in the position string is the ponder move. The engine can do what it wants to do, but after a 'ponderhit' command it should execute the suggested move to ponder on. This means that the ponder move sent by the GUI can be interpreted as a recommendation about which move to ponder. However, if the engine decides to ponder on a different move, it should not display any mainlines as they are likely to be misinterpreted by the GUI because the GUI expects the engine to ponder on the suggested move.
- `wtime`: white has x msec left on the clock.
- `btime`: black has x msec left on the clock.
- `winc`: white increment per move in mseconds if x > 0.
- `binc`: black increment per move in mseconds if x > 0.
- `movestogo`: there are x moves to the next time control, this will only be sent if x > 0, if you don't get this and get the wtime and btime it's sudden death.
- `depth`: search x plies only.
- `nodes`: search x nodes only.
- `mate`: search for a mate in x moves.
- `movetime`: search exactly x mseconds.
- `infinite`: search until the 'stop' command. Do not exit the search without being told so in this mode!"

Usage:

```
fish$go(
  searchmoves = NULL,
  ponder = NULL,
  wtime = NULL,
  btime = NULL,
  winc = NULL,
  binc = NULL,
  movestogo = NULL,
  depth = NULL,
  nodes = NULL,
  mate = NULL,
  movetime = NULL,
  infinite = FALSE
)
```

Arguments:

`searchmoves` A string with the only moves (separated by spaces) that should be searched

`ponder` Pondering move (see UCI documentation for more information)

`wtime` Time (in ms) white has left on the clock (if `movestogo` is not set, it's sudden death)

`btime` Time (in ms) black has left on the clock (if `movestogo` is not set, it's sudden death)

`winc` White increment (in ms) per move if `wtime` > `0`

`binc` Black increment (in ms) per move if `btime` > `0`

`movestogo` Number of moves to the next time control

`depth` Maximum number of plies to search

`nodes` Maximum number of nodes to search

`mate` Search for a mate in this number of moves

`movetime` Time (in ms) allowed for searching

`infinite` Whether to only stop searching when a `stop()` command is issued (makes function return instantly without any output)

Returns: A string with result of the search or `NULL` if `infinite` == `TRUE`

Method `stop()`**:** Stop calculating as soon as possible.

"Stop calculating as soon as possible, don't forget the 'bestmove' and possibly the 'ponder' token when finishing the search."

Usage:

```
fish$stop()
```

Returns: A string with the result of search or `NULL` if there was no search underway

Method `ponderhit()`**:** Tell the engine that the user has played the expected move.

"The user has played the expected move. This will be sent if the engine was told to ponder on the same move the user has played. The engine should continue searching but switch from pondering to normal search."

Usage:

```
fish$ponderhit()
```

Method `quit()`**:** Kill the engine

"Quit the program as soon as possible."

Usage:

```
fish$quit()
```

Method `print()`**:** Print information about engine process.

Usage:

```
fish$print(...)
```

Arguments:

`...` Arguments passed on to `print()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
fish$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`fish_find` *Find bundled Stockfish executable*

Description

This package comes bundled with Stockfish, an open source, powerful UCI chess engine. For more information about what Stockfish is, see the full documentation of this package by running `?fish`.

Usage

```
fish_find()
```

Index