pyksolve

Release 0.0.11

Mar 20, 2021

Contents:

CHAPTER 1

Installing pyksolve

1.1 Dependencies

pyksolve requires Python 3.6 or higher.

There are currently binary wheels available for Linux, macOSX and Windows on PyPI for various Python versions.

1.2 Installing binary wheels

Run the following command:

```
pip3 install --upgrade pyksolve
```

1.3 Installing from source

Otherwise the package can be built from source, which requires a C++ tool chain installed.

On Debian/Ubuntu based **Linux**, this command will install all dependencies you need to build pyksolve:

```
sudo apt install build-essential python3-dev
```

On **Windows**, this wiki page contains all necessary information to get a compiler installed.

This package is not, as of yet tested on **macOS**, although in CI it seems to build correctly for Python 3.8 .

1.3.1 PyPI

To install the source distribution from PyPI, run:

```
pip3 install --upgrade --no-binary pyksolve
```

1.3.2 GitHub

To install directly from the master branch of pyksolve, run the following command:

```
pip3 install --upgrade git+https://github.com/tcdude/py-klondike-solver.git
```

CHAPTER 2

Introduction

2.1 About pyksolve

`pyksolve` is a wrapper around Klondike-Solver using Cython.

2.2 Using pyksolve

2.2.1 Small usage sample

```
from pyksolve import solver

s = solver.Solitaire()
s.shuffle1(42)
s.reset_game()  # Needs to be called before a solve_* method runs!!

print(s.game_diagram())

result = s.solve_minimal_multithreaded(4)
if result == solver.SolveResult.SolvedMinimal:
    print('Found a solution:\n')
    print(s.moves_made())
else:
    print(f'No minimal solution found. SolveResult = "{repr(result)}"')
```

1. This code creates a `pyksolve.solver.Solitaire` instance `s` which is then shuffled with 42 as optionally specified random seed. *It is important, that `pyksolve.solver.Solitaire.reset_game()` is called before one of the solve_* methods is called.*
2. It prints out the game diagram after the shuffle, before trying to solve with a minimal solution using 4 hardware threads.

3. Finally it verifies whether a minimal solution was found and either prints the corresponding moves made or just what result code was received.

Further information is available in the *API Documentation*.

CHAPTER 3

API Documentation

3.1 pyksolve package

A wrapper around the Klondike-Solver.

3.1.1 Submodules

3.1.2 pyksolve.solver module

Provides the wrapped main function of "KlondikeSolver.cpp".

class `pyksolve.solver.`**`Solitaire`**

Bases: `object`

Wrapper around the Solitaire C++ class from Klondike-Solver.

`draw_count`

`int` -> Number of cards drawn for each draw move.

Setter: `int`

`foundation_count`

`int` -> Output of "FoundationCount()".

`game_diagram`(*self*)

Get the current game diagram in the default format.

Returns `str` -> The game diagram in the default format.

`game_diagram_pysol`(*self*)

Get the current game diagram in PySol format.

Returns `str` -> The game diagram in the PySol format.

`get_move_info`(*self*, *move_index*)

Move info as KlondikeSolver provides it.

Parameters **move_index** – `int` -> valid move index.

get_pysol(*self*)
Get the current card set in PySol format.

Returns `str` -> The card set in the PySol format.

get_solitaire(*self*)
Get the current card set.

Returns `str` -> The card set in the default format.

load_pysol(*self*, *card_set*)
Load a card set in the PySol format.

Parameters **card_set** – `str` -> The card set in the PySol format.

load_solitaire(*self*, *card_set*)
Load a card set in the default format.

Parameters **card_set** – `str` -> The card set in the default format.

moves_made(*self*)
Get a space delimited list of the moves made to solve.

Returns `str` -> The moves delimited by single spaces.

moves_made_count
`int` -> Output of "MovesMadeCount()".

moves_made_normalized_count
`int` -> Output of "MovesMadeNormalizedCount()".

reset_game(*self*, *draw_count=None*)
Calls the *ResetGame* method.

Parameters **draw_count** – `int` -> Number of cards drawn for each draw move.

shuffle1(*self*, *deal_number=-1*)
Calls the *Shuffle1* method.

Parameters **deal_number** – `int` -> Optional random seed.

Returns `int` -> Random seed used to shuffle.

shuffle2(*self*, *deal_number*)
Calls the *Shuffle2* method.

Parameters **deal_number** – `int` -> Random seed.

solve_fast(*self*, *two_shift=0*, *three_shift=0*, *max_closed_count=None*)
Attempts to find a fast but possibly not minimal solution.

Parameters

- **two_shift** – `Optional[int]` ->
- **three_shift** – `Optional[int]` ->
- **max_closed_count** – `Optional[int]` -> Maximum number of game states to evaluate before terminating. Defaults to *5,000,000*.

Returns `SolveResult` -> The result of the attempt.

solve_minimal(*self*, *max_closed_count=None*)
Attempts to find a minimal solution.

Parameters **max_closed_count** – `Optional[int]` -> Maximum number of game states to evaluate before terminating. Defaults to *5,000,000*.

Returns `SolveResult` -> The result of the attempt.

solve_minimal_multithreaded(*self*, *num_threads*, *max_closed_count=None*)

Attempts to find a minimal solution, using multiple threads.

Parameters

- **num_threads** – `int` -> Number of threads to use.
- **max_closed_count** – `Optional[int]` -> Maximum number of game states to evaluate before terminating. Defaults to *5,000,000*.

Returns `SolveResult` -> The result of the attempt.

class pyksolve.solver.**SolveResult**

Bases: `enum.Enum`

Solve result enum.

CouldNotComplete = -2

Impossible = 0

SolvedMayNotBeMinimal = -1

SolvedMinimal = 1

3.1.3 pyksolve.deferred module

Provides the DeferredSolver class that generates a number of solvable games for faster access to a solvable seed on demand.

class pyksolve.deferred.**DeferredSolver**(*draw_counts: Tuple[int, ...] = (1, 3)*, *cache_num: int = 5*, *threads: int = 3*, *max_closed: int = 1000000*, *seed: Optional[int] = None*)

Bases: `object`

Provides a cache of solved games, that is kept at a user defined number of games for each specified draw count. To properly clean up, call `DeferredSolver.stop()` when the *DeferredSolver* is no longer needed.

Parameters

- **draw_counts** – `Tuple[int, ...]` -> for which draw count a cache is generated. Defaults to *(1, 3)*.
- **cache_num** – `int` -> number of solvable games to cache at any time. Defaults to *5*.
- **threads** – `int` -> number of workers to run solvers. Defaults to *3*.
- **max_closed** – `int` -> max_closed argument to be passed to the used `pyksolve.solver.Solitaire.solve_fast()` method. Defaults to *1,000,000*.

> **Warning:** If you don't call `DeferredSolver.stop()`, your program might hang until terminated forcefully. After `DeferredSolver.stop()` was called, the class is defunct!

get_solved(*draw_count: int*) → Tuple[int, str, str]

Get a solved game from cache with the specified draw count.

Parameters **draw_count** – `int` -> valid draw count value as specified on init.

Returns Tuple of (seed, game_diagram before solved, moves_made).

stop()

Signals all threads to stop.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index