Knight’s Path Problem

Generated by Doxygen 1.7.6.1

Mon Aug 25 2014 21:34:12

Contents

Chapter 1

Knight Path Problem: This is problem to find certain paths that a knight can follow.

Author

Suresh Golconda

Description:

- Presently system implements 4 (of 5) problems, labelled as 'level'.
- level-1:Write a function that accepts a sequence of moves and reports whether the sequence contains only valid knight moves. It should also optionally print the state of the knight board to the terminal as shown above after each move. The current position should be marked with a 'K'.
- Level 2: Compute a valid sequence of moves from a given start point to a given end point.
- Level 3: Compute a valid sequence of moves from a given start point to a given end point in the fewest number of moves.
- Level 4: Now repeat the Level 3 task for this 32x32 board. Also, modify your validator from Level 1 to check your solutions. This board has the following additional rules: 1) W[ater] squares count as two moves when a piece lands there 2) R[ock] squares cannot be used 3) B[arrier] squares cannot be used AND cannot lie in the path 4) T[eleport] squares instantly move you from one T to the other in the same move 5) L[ava] squares count as five moves when a piece lands there

More information:

- Please read the README file for more information on how to compile, run, etc.

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

Namespace Documentation

4.1 knight_t Namespace Reference

Classes

- struct cell_ind_t

  To store the index of a cell on the board.
- struct cell_t

  Used while storing a cell for path exploration.
- class board_t
- class search_path_t

Typedefs

- typedef std::vector$<$ cell_ind_t $>$ **path_t**

Functions

- void set_board_cost_8x8 (board_t &board_)
- void set_board_cost_16x16 (board_t &board_)
- void set_board_cost_32x32 (board_t &board_)
- void display_path_on_board (path_t const &path_, int dim_size_)
- void scan_board_diagram (std::string filePath_, board_t &brd_, int DIM_SIZE_)
- bool scan_path_string (std::string indata_, int &dim_size_, path_t &path_)
- bool scan_dim_src_dest (std::string indata_, int &dim_size_, cell_ind_t &src_, cell_ind_t &dest_)

Variables

- unsigned const int **BLOCKED** = INT_MAX

- unsigned const int **WATER** = (INT_MAX -1)
- unsigned const int **ROCK** = (INT_MAX - 2)
- unsigned const int **BARRIER** = (INT_MAX - 3)
- unsigned const int **TELEPORT** = (INT_MAX - 4)
- unsigned const int **LAVA** = (INT_MAX - 5)
- unsigned const **KNIGHT_C** = (INT_MAX -6)
- unsigned const int **MIN_VALID_VALUE** = (INT_MAX - 7)

4.1.1 Detailed Description

File to contain utility functions.

Author

Suresh Golconda

4.1.2 Function Documentation

4.1.2.1 void **knight_t::display_path_on_board** (path_t const & *path_*, int *dim_size_*)

Display the given path, on the board, by numbering the cells along the path.

4.1.2.2 void **knight_t::scan_board_diagram** (std::string *filePath_*, board_t & *brd_*, int *DIM_SIZE_*)

Utility function to read the input board_diagram file and mark, given board with respective cells of {water, rock, barrier, teleport, lava}

4.1.2.3 bool **knight_t::scan_dim_src_dest** (std::string *indata_*, int & *dim_size_*, cell_ind_t & *src_*, cell_ind_t & *dest_*)

Utility function to scan, board's dimension, src, and destination cell information from given string.

4.1.2.4 bool **knight_t::scan_path_string** (std::string *indata_*, int & *dim_size_*, path_t & *path_*)

Utility function to scan given string for sequence of points along the path.

4.1.2.5 void **knight_t::set_board_cost_16x16** (board_t & *board_*)

Utility function for testing sample board configuration in a 16x16 board

4.1.2.6 void knight_t::set_board_cost_32x32 (board_t & *board_*)

Utility function for testing sample board configuration in a 32x32 board

4.1.2.7 void knight_t::set_board_cost_8x8 (board_t & *board_*)

Utility function for testing sample board configuration in a 8x8 board

Chapter 5

Class Documentation

5.1 knight_t::board_t Class Reference

```
#include <board.H>
```

Public Member Functions

- void display_board ()
- void display_move (cell_ind_t const &move_)
- void display_path (path_t const &moves_)
- void set_teleport_cells (int teleport_xi_[ ], int teleport_yi_[ ])
- board_t (int dim_size_)
- void reset (int value_)
- bool on_board (cell_ind_t const cell_)
- int value (int xi, int yi)
- void set (int xi, int yi, int value, int parent_xi=-1, int parent_yi=-1)
- cell_t & get (int xi, int yi)
- std::vector< int > const & get_teleport_x ()
- std::vector< int > const & get_teleport_y ()
- bool knight_neighbor (cell_ind_t const &c1, cell_ind_t const &c2)
- bool teleport_points (cell_ind_t const &c1, cell_ind_t const &c2)
- bool check_valid_path (path_t const &path_, bool display_)

5.1.1 Detailed Description

Class to contain check board configuration. It is initialized with DIM_SIZE, size of the chess board in one dimension.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 knight_t::board_t::board_t (int *dim_size_*) [inline]

Parameters

dim_size_,:	dimension of the board

5.1.3 Member Function Documentation

5.1.3.1 bool board_t::check_valid_path (path_t const & *path_,* bool *display_*)

Returns true if sequence of points in path is valid knight moves and falls within the board.

5.1.3.2 void board_t::display_board ( )

Display board

5.1.3.3 void board_t::display_move (cell_ind_t const & *move_*)

Display board with given move marked on it

5.1.3.4 void board_t::display_path (path_t const & *moves_*)

Display the board for each move

5.1.3.5 cell_t& knight_t::board_t::get (int *xi,* int *yi*) [inline]

Returns reference to the cell at (xi, yi)

5.1.3.6 std::vector<int> const& knight_t::board_t::get_teleport_x ( ) [inline]

Returns constant reference to vector of x coordinates of teleport cells.

5.1.3.7 std::vector<int> const& knight_t::board_t::get_teleport_y ( ) [inline]

Returns constant reference to vector of y coordinates of teleport cells.

5.1.3.8 bool knight_t::board_t::knight_neighbor (cell_ind_t const & *c1,* cell_ind_t const & *c2*) [inline]

Checks if the pair of given cells are one knight step away.

5.1.3.9 bool **knight_t::board_t::on_board** (**cell_ind_t** const *cell_*) [inline]

Returns true if the given cell falls within the board

5.1.3.10 void **board_t::reset** (int *value_*)

Resets all the cells on the board with given value

5.1.3.11 void **knight_t::board_t::set** (int *xi*, int *yi*, int *value*, int *parent_xi* = -1, int *parent_yi* = -1) [inline]

Sets the value and parent node for given cell

5.1.3.12 void **board_t::set_teleport_cells** (int *teleport_xi_[]*, int *teleport_yi_[]*)

Sets given arra of points as teleport cells

5.1.3.13 bool **knight_t::board_t::teleport_points** (**cell_ind_t** const & *c1*, **cell_ind_t** const & *c2*) [inline]

Return true if points c1, c2 are both teleport points

5.1.3.14 int **knight_t::board_t::value** (int *xi*, int *yi*) [inline]

Returns the value of the given cell (xi, yi)

The documentation for this class was generated from the following files:

- board.H
- board.C

5.2 knight_t::cell_ind_t Struct Reference

To store the index of a cell on the board.

```
#include <board.H>
```

Public Attributes

- int **xi**
- int **yi**

5.2.1 Detailed Description

To store the index of a cell on the board.

The documentation for this struct was generated from the following file:

- board.H

5.3 knight_t::cell_t Struct Reference

Used while storing a cell for path exploration.

```
#include <board.H>
```

Public Attributes

- int **value**
- int **parent_xi**
- int **parent_yi**

5.3.1 Detailed Description

Used while storing a cell for path exploration.

The documentation for this struct was generated from the following file:

- board.H

5.4 knight_t::search_path_t Class Reference

```
#include <search_path.H>
```

Classes

- struct **intermediate_steps_t**

  inter mediate steps between knight's single move
- class **node_t**

Public Member Functions

- search_path_t (unsigned dim_size_, board_t ∗base_board_, bool debug_=false)
- int new_cost (const node_t &nd, int nxi, int nyi)
- bool move_blocked (const node_t &nd, intermediate_steps_t const ∗path)

- bool get_path (int sxi_, int syi_, int exi_, int eyi_, path_t &path_)
- void trace_path (node_t const &dest, node_t const &src, board_t &search_bd, path_t &path_)
- void set_teleport_nav (node_t const &nd, board_t &search_bd, std::priority_-queue< node_t > &pq)
- intermediate_steps_t const ∗ move_intersteps (int delx, int dely)
- void display_step_path (intermediate_steps_t const ∗path_)

5.4.1 Detailed Description

Class to implement exploration of the chess board to find shortest path from given source cell to given destination cell.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 knight_t::search_path_t::search_path_t (unsigned *dim_size_*, board_t ∗ *base_board_*, bool *debug_* = `false`) `[inline]`

Parameters

*debug_*	if set will print additional debug information. debug_ parameter is option, is by default taken as false.

5.4.3 Member Function Documentation

5.4.3.1 void knight_t::search_path_t::display_step_path (intermediate_steps_t const ∗ *path_*) `[inline]`

Prints the intermediate cells of a knight's single step

5.4.3.2 bool search_path_t::get_path (int *sxi_*, int *syi_*, int *exi_*, int *eyi_*, path_t & *path_*)

Implements the Best-First-Search for finding a path from (sx_i,syi_) to (exi_, eyi_). -Reference: `http://en.wikipedia.org/wiki/Best-first_search`

5.4.3.3 bool search_path_t::move_blocked (const node_t & *nd*, intermediate_steps_t const ∗ *path*)

Return true if the path blocked by barrier. Else returns false.

5.4.3.4 search_path_t::intermediate_steps_t const ∗ search_path_t::move_intersteps (int *delx*, int *dely*)

Return the 2 intermediate (adjacent) steps in one step of knigh

5.4.3.5 int **search_path_t::new_cost** (const node_t & *nd*, int *nxi*, int *nyi*)

Returns the cost for neighbor cell when traversing from nd.

5.4.3.6 void **search_path_t::set_teleport_nav** (node_t const & *nd*, **board_t** & *search_bd*, std::priority_queue< node_t > & *pq*)

Marks the other end of teleport cell with equal cost and adds to exploration list.

5.4.3.7 void **search_path_t::trace_path** (node_t const & *dest*, node_t const & *src*, **board_t** & *search_bd*, path_t & *path_*)

Traces the path from dest to source, to extract sequence of cells (path)

The documentation for this class was generated from the following files:

- search_path.H
- search_path.C