Sliding Block Puzzle AI

Project Overview

This project includes the implementation for a sliding block puzzle, sample sliding block puzzles in `*.txt` format, and several algorithms for efficiently finding solutions to a generic puzzle. The solutions include A*, breadth first search, depth first search, and iterative deepening search.

Running this program

To run this program, run `main.py` on its intended platform, `Python 2.7`. `main.py` takes two inputs, the name of a text file containing the sliding block puzzle to be attempted, and a string denoting which algorithm should be used.

There are several example puzzles included with this program that it can be tested on, please see the 'puzzles' folder.

The options for the second argument are as follows: `bfs` : use breadth first search to solve the puzzle `dfs` : use depth first search to solve the puzzle `id` : use iterative deepening search to solve the puzzle `astar` : use an A* search to solve the puzzle

I've included an example command line executions of the code: Unix: "python `main.py` puzzles/puzzle2.txt 'bfs' "

Output

`main.py` outputs a solution path and some search details after the solution to a given puzzle is found. It also prints the final board state to the console. Each of the algorithms in this project build a search tree while searching for the solution. The "nodes explored" output refers to the total number of nodes (as well as the total number of board states) that have been added to the search tree.

The "solution length" output refers to the number of moves it would take to reach the final solution from the initial state.

The path is shown by listing the moves necessary to go from the initial state to the final state. Moves are displayed as a block number and the direction that block is moved. Please note, the block number is the one assigned to the block after the state has been normalized. As a result, it can be a little tricky to follow the progression of the puzzle move-by-move.

Mechanics

Sliding block puzzles must be loaded from a `*txt` file. Each text file must only contain one puzzle. The first line of the file must include the dimensions of the puzzle board. The following lines depict the actual puzzle. `-1`: denotes the goal space, when this is covered by the solution block the puzzle

is complete. `0` : denotes an empty space `1` : denotes a wall, which can not be occupied `2` : denotes the solution block, which must be moved off of the board `3+`: denotes an obstacle block, they are only distinguishable by having different numbers

Individual moves can be applied to change the position of one block by one row or column. The entire block must be moved simultaneously, and all destination spaces must be empty prior to the move.

Architecture

Algorithm file

The `algorithms.py` file includes only the core algorithms, all helper functions are located in a separate file. The algorithm file includes: A*, breadth first search, depth first search, and iterative deepening search. It also includes a heuristic function which is utilized by the A* algorithm.

Driver file

The `main.py` file contains the main function utilized by this project. It has two inputs, one to specify the `*.txt` file to be used as the puzzle, and the other to specify the search strategy to utilize. Sliding Block Puzzle AI

Helper Function file

The `helperFunct.py` file includes helper functions and the sliding block mechanics which are used by the `algorithms.py` and `main.py` files.