COMSW4701: Artificial Intelligence

Assignment 2

William Falk-Wallace (wgf2104)

The submitted zip folder contains my submission for AI Assignment 2. It has a command line prompt which requests level file (as specified in the assignment document) as well as search type and whether or not to display statistics. Those parameters may also be specified as command line arguments.

1. How To Run:
    1. make sure `runtests.sh` is executable: `chmod a+x runtests.sh`
    2. run `./runtests.sh`

    Alternatively

    1. run `make`
    2. run `java SokoTest "<level-file>" <int> <y/n>`

        Where

        - `<level-file>` is a text file containing a valid level
        - `<int>` is an integer corresponding to a search type, listed below
        - `<y/n>` indicates whether or not to display statistics

        (ex. `make`; `java SokoTest "levels/0.txt" 1 y`)

    Alternatively

    1. run `make`
    2. run `java SokoTest`
2. Searches Implemented (with corresponding CLI arguments):
    1. BFS
    2. DFS
    3. UCS
    4. Greedy Best First Search (with open goals heuristic)
    5. Greedy Best First Search (with Manhattan distance heuristic)
    6. A* Search (with open goals heuristic)
    7. A* Search (with Manhattan distance heuristic)
3. Heuristics Implemented:
    1. **Manhattan Distance to Goal:** computes the Manhattan Distance (sum of horizontal and vertical distance) from all boxes to the goal nearest to them.
    2. **Open Goals:** computes the number of open (without a box covering them) goals present in the state.

    Both of these heuristics are admissible and consistent, given the cardinal-motion

constraint imposed on the warehouse keeper (sokoban agent).

4. Files Included:
    1. `runtests.sh`: A shell script to run all 21 tests (explained below) and produce a logfile containing the results.
    2. `Terminal Saved Output.txt`: The results of 21 tests (3 puzzles, below, run on each of 7 search types).
    3. `Makefile`: Makefile for compiling and cleaning up the porgram.
    4. `State.Java`: A representation of a *Sokoban State*; the State Class encapsulates a matrix and string representation of the layout of the warehouse, the path to reach itself from the initial state, the path-cost to be reached, as well as functions to carry out the transition from one state/layout to the next, given a direction of motion of the warehouse keeper.
    5. `STree.java`: A representation of a *Sokoban Search Tree*; the STree Class encapsulates a traversal graph that is dynamically computed depending on the specific search run. It also defines the search functions themselves.
    6. `LevelLoader.java`: This class handles reading in a formatted level-file and casting it to a State.
    7. `SokoTest.java`: Handles the UI and inititialization of LevelLoader and STree, as well as calling the searches and returning the results.
    8. `ManhDist.java`: A Comparator based on the Manhattan Distance heuristic; It is used in the Greedy search to determine preferential traversal ordering in the PriorityQueue implementation.
    9. `OpenGoals.java`: A Comparator based on the Manhattan Distance heuristic; It is used in the Greedy search to determine preferential traversal ordering in the PriorityQueue implementation.
    10. `StarMD.java`: A Comparator based on the Manhattan Distance heuristic; It is used in the A* search to determine preferential traversal ordering in the PriorityQueue implementation.
    11. `StarOG.java`: A Comparator based on the Manhattan Distance heuristic; It is used in the A* search to determine preferential traversal ordering in the PriorityQueue implementation.
    12. `levels`: A collection of sample levels taken from Professor Voris' file on Courseworks and used to test search functionality. These are the initial-layouts used to perform the tests listed in `Terminal Saved Output.txt`.
        1. `0.txt`
        2. `1.txt`
        3. `3.txt`