Sokoban Puzzle Solver

Name: Nate Brennand
UNI: nsb2142
Class: COMS 4701

Running/Testing

```
# to see all options
python main.py --help
```

To run the tests, use `python main.py --tests`. This will run each search with 3 maps provided by Voris.

To run an indivual puzzle, use `python main.py path/to/the/puzzle.txt`. All puzzle provided by Jon Voris are located in the `sokoban_boards/` directory.

To compare only GBFS and A* use the `--informed` flag.

- a copy of the data I gathered on my local machine in `standard_test_output.txt`
- a comparison of GBFS & A* on more complicated puzzles is in `informed_test_output.txt`

Overview

The Sokoban puzzle board is represented by the `Board` object (`board/board.py`). Each item on the board (wall, goal, box, etc) is represented with a `Position` object (`board/position.py`). Using a `Position` object allowed the `Board` object code to be much clearer with overloaded methods. All `Position` objects in `Board` are held in sets. This makes the code very easy to read and relatively fast and allowed easy comparison of board states caculating a hash based on the items on the board.

The five search algorithms are located in the `search` directory. In each search (including DFS), nodes are checked for duplication before being added to the queue. This makes the DFS faster than a more standard and basic implementation would be. Two similar priority queues are also in `searches/` for use by Greedy Best First and A* searches.

Sokoban maps (examples in `sokoban_boards/`) are loaded by `map_loader.py`. In `main.py` maps are loaded before being passed to each of the searches. Reports are then compiled with data returned by the searches.

Thoughts

BFS, DFS and UCS all have similar runtimes. The best for each map seems to depend more on the shape than anything else. Due to their nature their runtimes skyrocket as the map size increases.

Both GBFS and A *perform significantly better than the uninformed searches. My GBFS implementation tends to beat my A* on the simpler puzzles I used for testing. When I turned off the other searches to compare just these two and increased the complexity A* began to outperform BGFS as would be expected. Examples of comparisons between just these two searches are located in `informed_test_output.txt`

Notes

DFS

Nodes are checked for duplicity before being added to the queue. This makes my DFS run faster than more primitive DFS because loops are prevented from occuring.

Heuristic

The same heuristic was used for both Greedy Best First and A* Searches

I used a naive heuristic using Manhattan style distances. The returned value is the sum of estimates for the player's distance to the nearest out-of-place blocks and and the distance of out-of-place blocks to goals.

- Manhattan distance between player and the nearest block not in a goal
- Manhattan distance between each block w/o a goal and the nearest goal
    - Mutiple blocks "aiming" at the same goal is not adjusted for

Because multiple blocks can "aim" at the same goal and player maneuvering is not accounted for, this is an **admissible heuristic**. The number of steps to finish from the next state will never be less than the estimate by this heuristic.