Bryan Jensen
COMP-255
February 23, 2013

The 8-Puzzle using Best First & A* Searches

Program Summary:

This program incorporates functions from the my_searches and classes from the BoardClass modules to solve the classic 8-puzzle game. Once a board is constructed, it is passed to the search function, which returns the solution steps. The BoardClass class incorporates much of the needed functionality of the searching such as the heuristics and the ability to make children for a tree, while the search function itself simply implements these in the specified algorithm.

Contained in this report is an overview of the actual implementation of the various components of this program along with a history of applied optimizations. Examples will be given of the program's output, with timing for each of the inputs contained in the input set. Various tables will catalog the various improvements made to the algorithm of the program.

Methods:

The BoardClass is initialized with no parameters. The actual board of object (a list of lists) can then be randomly initialized with the initialize_board method, which takes the width of the board and other, optional parameters such as the type of heuristic (also settable in the constructor and/or with the set_type method), the method of generating the board (random or not) and the difficulty of the non-random board. The board object is then passed into the search function from my_searches, with the output from the function being stored for printing.

Various test boards are burned in, with the labeled difficulty of 0 (Easy), 1 (Less Easy) and 2 (Medium) based on the length of time for the solution. The board can also be randomly generated and all boards are checked for validity according to the given algorithm regarding inversions.

Heuristic algorithms used are the Displaced Tiles heuristic, simply totaling all tiles out of place, and the Manhattan Distance heuristic, which counts the number of direct moves it would take to move every piece into place ignoring the rules of the game and physics.

Searching algorithms used include the Best First Search algorithm, implementing a priority queue (implicitly using the python built-in heapq, a balanced tree) as the queue for nodes, and a list for the already visited nodes. The A* search algorithm was also used, implementing once again a priority queue and a list.

Sample Outputs:

Heuristic Level: 10
```
---------------
| 3 | 2 | 5 |
---------------
| 4 | 1 | 8 |
---------------
| 6 | 0 | 7 |
---------------
```
Heuristic Level: 10
```
---------------
| 3 | 2 | 5 |
---------------
| 4 | 1 | 8 |
---------------
| 6 | 7 | 0 |
---------------
```
Heuristic Level: 8
```
---------------
| 3 | 2 | 5 |
---------------
| 4 | 1 | 0 |
---------------
| 6 | 7 | 8 |
---------------
```
Heuristic Level: 6
```
---------------
| 3 | 2 | 0 |
---------------
| 4 | 1 | 5 |
---------------
| 6 | 7 | 8 |
---------------
```
Heuristic Level: 4
```
---------------
| 3 | 0 | 2 |
---------------
| 4 | 1 | 5 |
---------------
| 6 | 7 | 8 |
---------------
```
Heuristic Level: 4
```
---------------
| 3 | 1 | 2 |
---------------
| 4 | 0 | 5 |
---------------
| 6 | 7 | 8 |
---------------
```
Heuristic Level: 2
```
---------------
| 3 | 1 | 2 |
---------------
| 0 | 4 | 5 |
---------------
| 6 | 7 | 8 |
---------------
```
Heuristic Level: 0
```
---------------
| 0 | 1 | 2 |
---------------
| 3 | 4 | 5 |
---------------
| 6 | 7 | 8 |
---------------
```

Results:

The test suite was run through with every combination of board, heuristic and search to obtain an idea of how well the different solutions ran and the results thereof. The data is shown in table 1.

Input:	Search Type	Board Heuristic	Moves in Solution	Run Time	Max Queue Size	Notes:
random= False,diff =1	BestFS	D'ed Tiles	8	0.049		
random= False,diff =1	BestFS	M Distance	8	0.044		
random= False,diff =1	A*	D'ed Tiles	N/A	N/A		A* Algorithm doesn't seem to handle Displaced Tiles Heuristic well. Runtime was terminated.
random= False,diff =1	A*	M Distance	8	0.051		
random= False,diff =2	BestFS	D'ed Tiles	64	7.769		First time we notice much time is being spent in BoardClass.__eq__
random= False,diff =2	BestFS	M Distance	42	0.144		
random= False,diff =2	A*	D'ed Tiles	N/A	N/A		See previous A*, D'ed Tiles run
random= False,diff =2	A*	M Distance	26	31.16		Time primarily taken up by __eq__ and range.

Table 1: Simple test suite of all configurations of burnt-in boards, search algorithms and board heuristics. Max queue sizes weren't recorded for these runs so column is grayed out.

At this point a major improvement was made off of the data from the first test run, with a hash being made of any board for comparison in the __eq__ method. That reduced a run taking 30 seconds to one taking 8 seconds. The program was then put through 25 runs of random boards, Manhatten

Distance only, in A* and Best First Search each, comparing the runtimes, average solution length and max queue size.

Input:	Search Type	Board Heuristic	Avg Moves	Total Runtime	Max Queue Size	Notes:
random= True	BestFS	M Distance	49	0.7	328	Average number of moves varies a lot, even with 25 runs.
random= True	A*	M Distance	26	126.2	1324	Runtime much longer, but complete and optimal.

Table 2: 25 runs of A* and Best First Search, comparing the different Big O's of the two algorithms.

Conclusion:

Best First Search and A* both have their pros and cons, and the tables of data above demonstrate that nicely. The main pro of A* is the thoroughness and the finding of the optimal solution, but Best First Search does command a major lead when it comes to speed.