Project 1 CS 170 Introduction to Artificial Intelligence

Thomas Shiels Instructor: Dr. Eamonn Keogh

ID 861151979

tshie001@ucr.edu

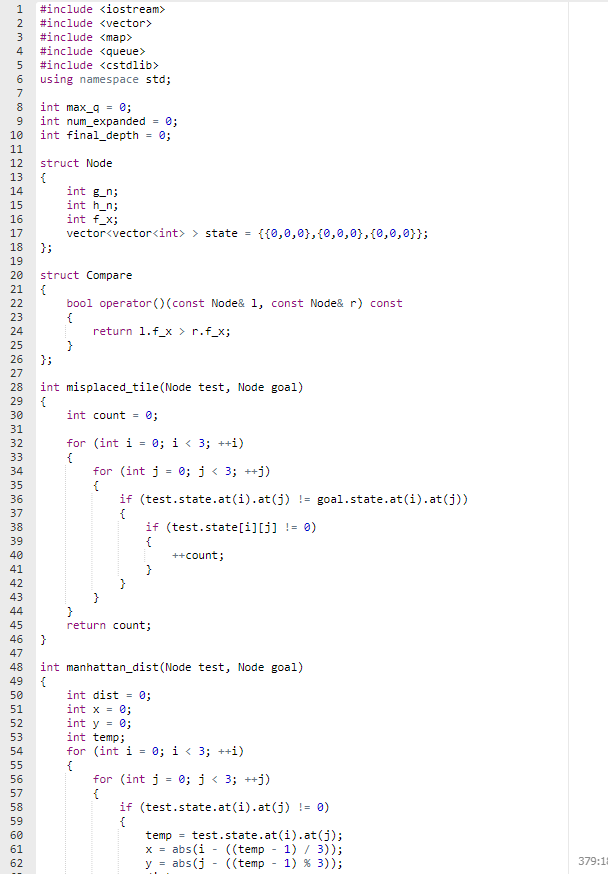
8-November-2018

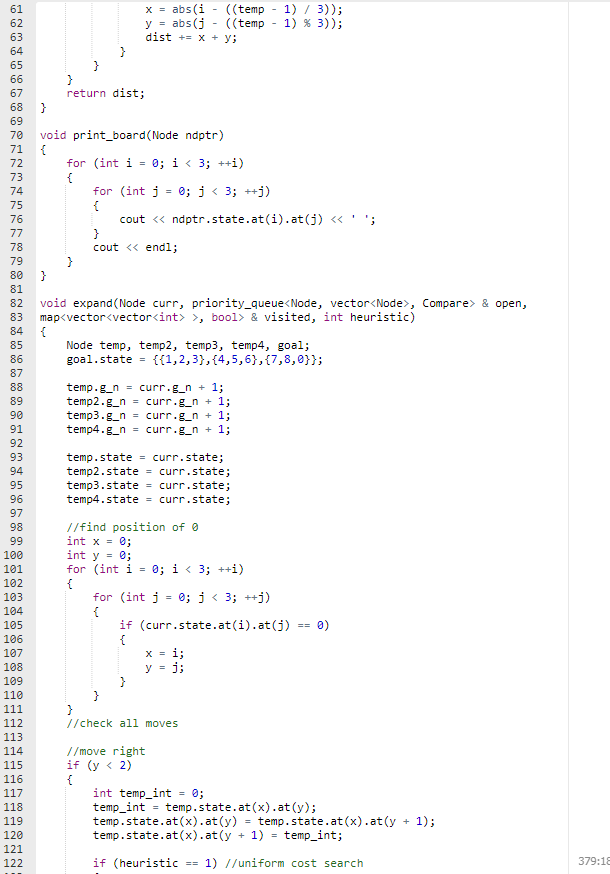
In completing this assignment I consulted:

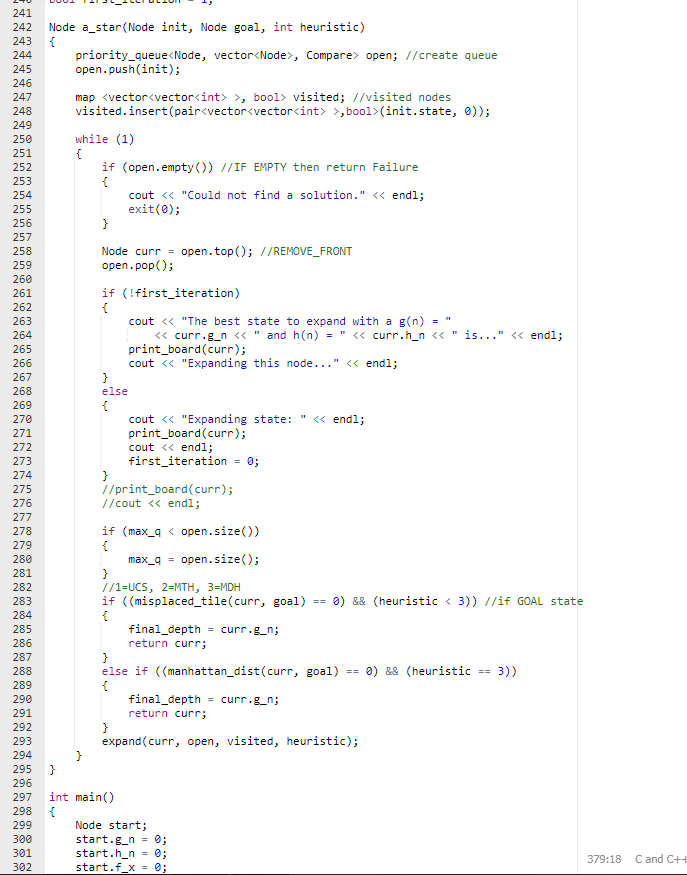
* For help with using a priority queue: http://www.cplusplus.com/reference/queue/priority\_queue/
* For help with using a custom comparator with a priority queue: https://stackoverflow.com/questions/16111337/declaring-a-priority-queue-in-c-with-a-custom-comparator
* The Blind Search and Heuristic Search slides from lecture.
* For clarification on the A\* algorithm: https://algorithmsinsight.wordpress.com/graph-theory-2/a-star-in-general/
* Further clarification on the A\* algorithm: https://www.cs.princeton.edu/courses/archive/fall15/cos226/assignments/8puzzle.html
* For learning how to use the <map> data structure in C++: http://www.cplusplus.com/reference/map/map/
* For learning how to search for elements in the <map> data structure: http://www.cplusplus.com/reference/map/map/find/
* For learning how to insert an element into the <map> data structure: http://www.cplusplus.com/reference/map/map/insert/
* To learn how to utilize the calculation for the Manhattan distance heuristic for 8-puzzle: https://stackoverflow.com/questions/29470768/finding-target-coordinates-for-manhattan-distance-in-8-puzzle-c
* For generating random 8-puzzles: http://mypuzzle.org/sliding

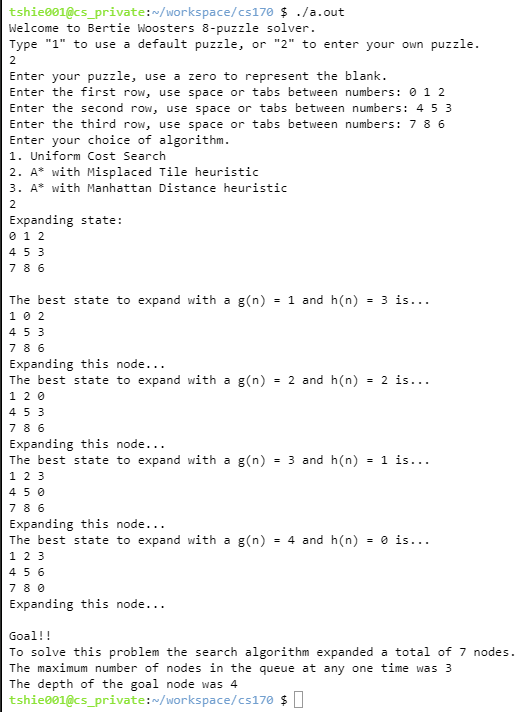
All important code is original. Unimportant subroutines that are not completely original are...

* The function "abs" from <cstdlib>
* All subroutines from <vector> standard library
* The "priority\_queue" data type from <queue> library
* All subroutines from the <map> standard library









CS170: Project 1 Write Up

Thomas Shiels, SID 861151979

Introduction

This purpose of this first project in Dr. Eamonn Keogh's Introduction to Artificial Intelligence course is to learn about solving the 8-puzzle game using a search tree, along with several different heuristics. This report will detail my findings in solving the game using Uniform Cost Search, Misplaced Tile, and Manhattan Distance heuristics. The programming language I used was C++, as this is the language I am most comfortable with.

Uniform Cost Search

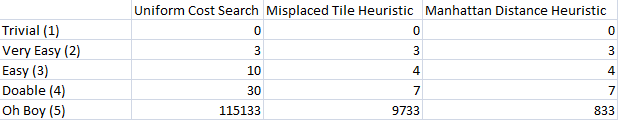
The Uniform Cost Search heuristic is the same as A\*, with the only metric being the depth of the node in the search tree. In effect, h(n) becomes hardcoded to zero, and the search degenerates into Breadth First Search. There are no weights to expansions, and each node has a cost of 1.

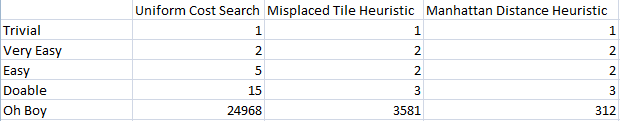
Misplaced Tile Heuristic

The first true heuristic in this project was the Misplaced Tile heuristic, which counts the number of tiles that are in the wrong position. This gives a vague idea of how close we are to solving the puzzle. The closer to solving the problem we are, the lower the score. As opposed to Uniform Cost Search, this heuristic is added to the cost of the current node, giving each edge of the tree a distinct weight. The nodes at a particular depth are inserted into the queue in ascending order of cost.

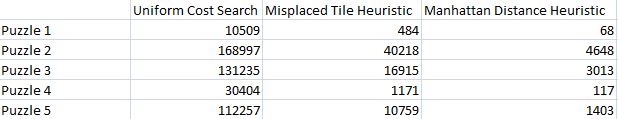
Manhattan Distance Heuristic

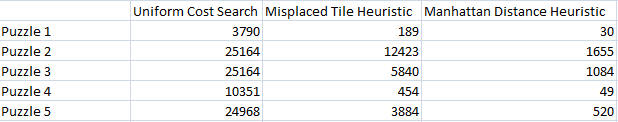
The Manhattan Distance Heuristic is the most sophisticated of the heuristics used in this assignment. It is similar to the Misplaced Tiles heuristic, but in addition to noting all misplaced tiles, it takes into account the distance from its correct position. The final output of the heuristic is the sum total of the distances of all the misplaced tiles.





Random Puzzles





Conclusion

There was a variety of difficulty in the puzzles, ranging from trivial to extremely difficult. The trivial puzzle was an already solved puzzle, where no nodes needed to be expanded, and where there was only ever one element in the queue at any given time. The Manhattan Distance heuristic was by far the most effective of all. In the more difficult puzzles it made most of the difference, solving the puzzle faster by orders of magnitude. The second most effective heuristic was the Misplaced Tiles heuristic. Uniform Cost Search performed the worst by far with having no way to discern a more optimal path. While a heuristic can provide a huge benefit, it is important to make sure your heuristic is a good fit for the problem at hand.