Christian Lo (clo060), Ethan liu (eliu058),
Calvin Trujillo (ctruj011) ,
Krishaan Patel (kpate135)
Section 001

CS170 - Project 1 Report

Introduction:

The 8-puzzle problem is a perfect example of how different algorithms will go about solving a certain problem. For this project, we are tasked to implement two different algorithms, one with two different variants which will show us the different time complexities they will take to solve. The two different algorithms that we have used will be the Uniform Cost Search Algorithm and the A* algorithm with misplaced tile/euclidean distance heuristics.

Data Recorded:

Maximum Queue Size

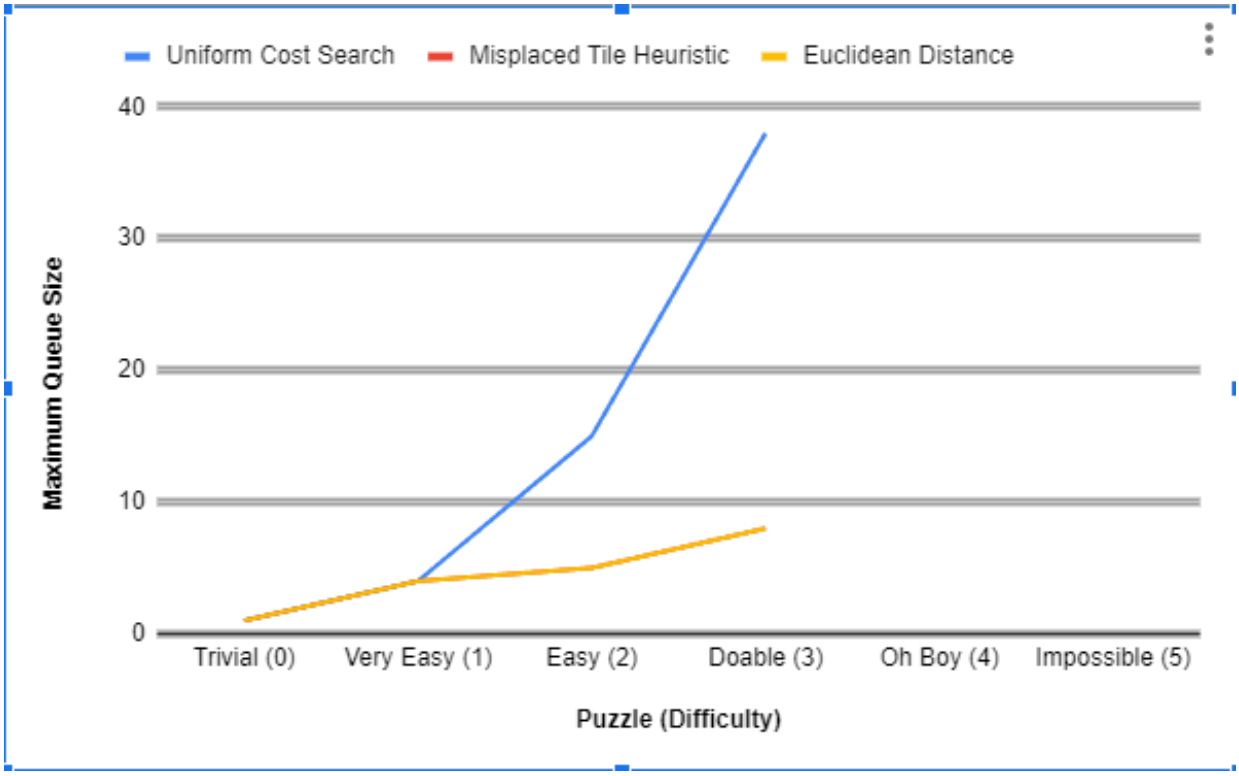
	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	**1**	**1**	**1**
Very Easy (1)	**4**	**4**	**4**
Easy (2)	**15**	**5**	**5**
Doable (3)	**38**	**8**	**8**
Oh Boy (4)	**(took WAYYY long)**	**(took too long)**	**(took too long)**
Impossible (5)	**terminated**	**terminated**	**terminated**

Number of Nodes Expanded

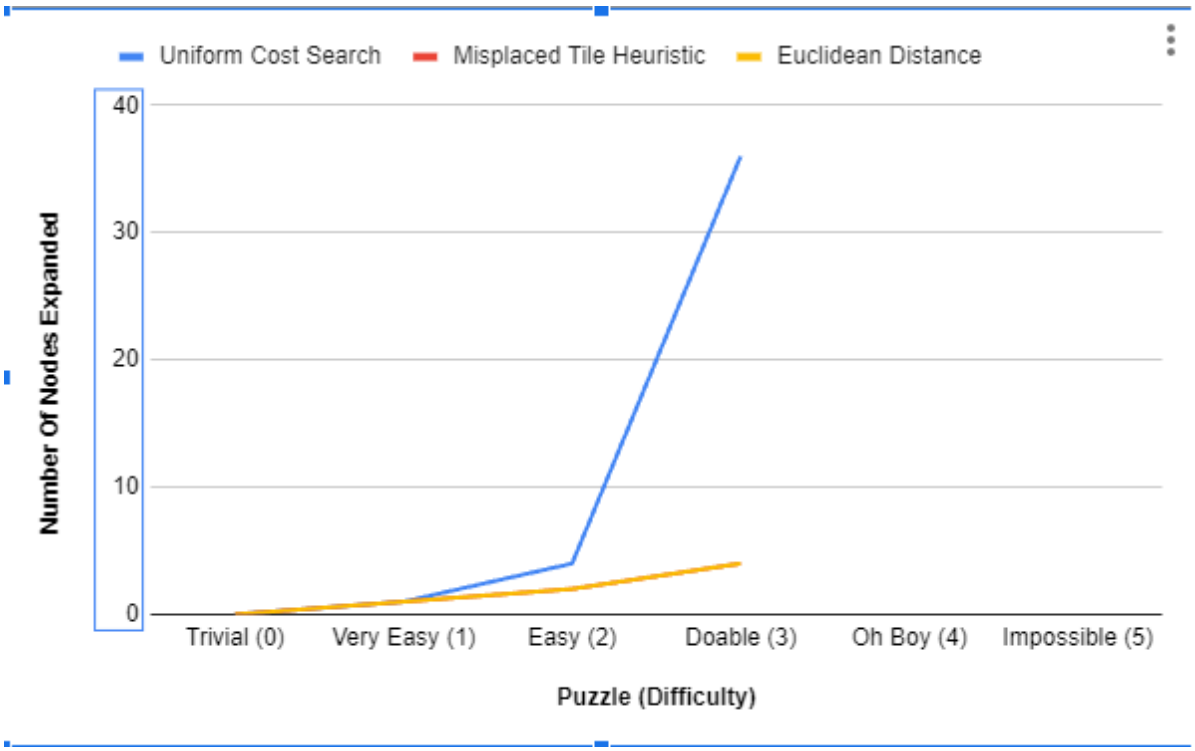
	Uniform Cost Search	Misplaced Tile Heuristic	Euclidean Distance Heuristic
Trivial (0)	**0**	**0**	**0**
Very Easy (1)	**1**	**1**	**1**
Easy (2)	**4**	**2**	**2**
Doable (3)	**36**	**4**	**4**
Oh Boy (4)	**(took too long)**	**(took to long)**	**(took to long)**
Impossible (5)	**terminated**	**terminated**	**terminated**

Diagrams:

Maximum Queue Size



Number of Nodes Expanded



Challenges:

Some of the main challenges in our project was configuring the priority queue data structure, in which we would be constantly getting errors in our trial due to not having the correct parameters being set. The libraries themselves were also an extreme task to understand and process. Knowing what library contains which defines parameters or what libraries to use for .cpp or .h pseudocode. It was very difficult to keep track of every node class with its vectors.

There were also issues with getting the libraries to work with one another without having the problem of them overriding similar functions they shared between each other. This was an issue as the compiler we were using g++ was not used to and stopped us from being able to fully test out the program.

When trying to determine how to establish the priority queue there were some difficulties. Because we wanted to sort by the lowest cost, that meant we had to establish the priority queue as a Min Heap which took a lot of trial and error to figure out. Furthermore, coming up with the behavior of replacing a node within the priority queue was challenging as there is no C++ STL function for replacing elements. Thus, we had to think carefully about it and come up with our own function to handle this behavior by populating a new frontier with the new node we want to replace the old node with.

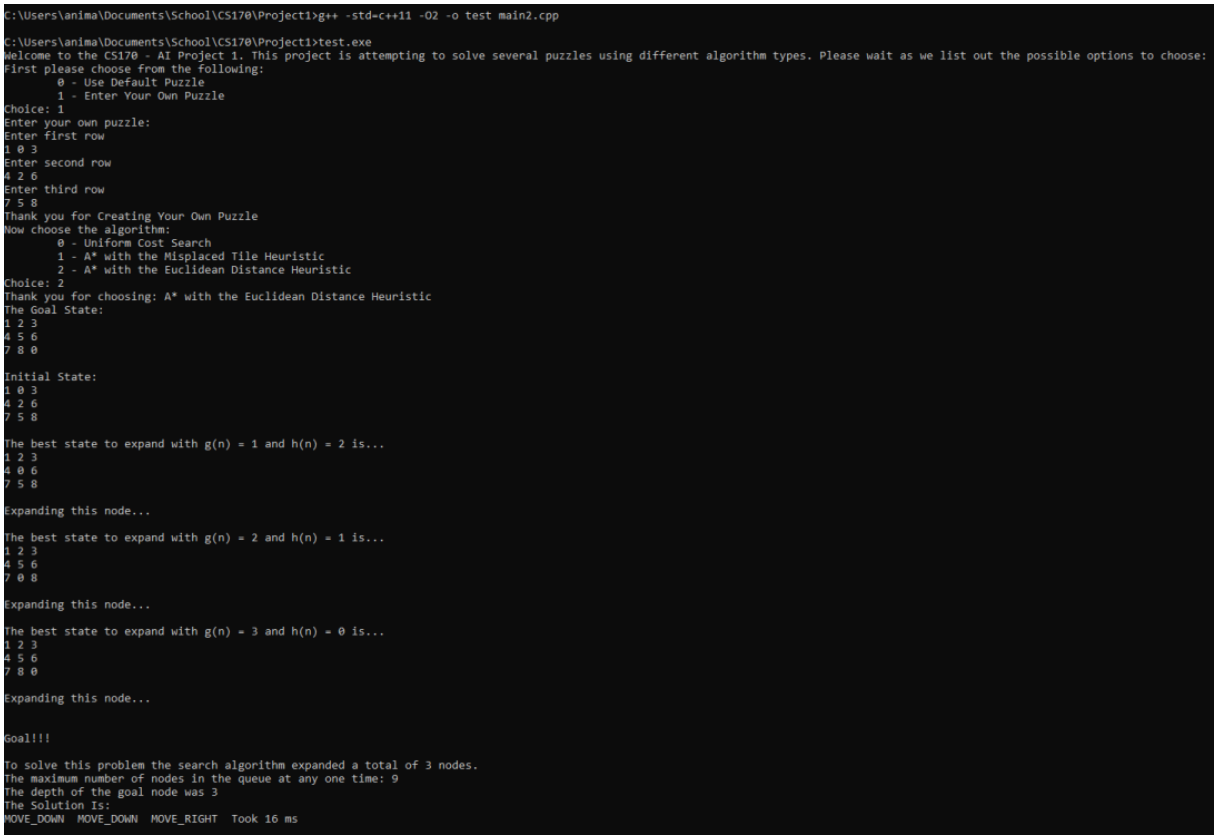
Solutions:

Some of the solutions that we had come up with were changing the layout of certain commands, changing the method of constructing the functions, and our logic changed quite a bit when coding the program. The changing of the layout helped a lot more than initially thought since it allowed us to not get the overriding other definition error. The compiler was able to full complete its work without any hassle after that was done. The alteration of our construction method was also something we needed to do as some portions of the code would not work correctly without it. We were not able to switch between what operations we were able to access at any given moment.

Conclusions:

Overall, we found that certain algorithms were more efficient at solving certain types of problems then others were. For example, the uniform cost search was extremely inefficient when dealing with larger data packs than Misplaced or Euclidean Distance Heuristic Search. We could establish this through their time complexities and nodes being expanded. We also learned that there is a difference between how you calculate the heuristic and if you try to use the default one of misplaced. When trying to find smaller heuristics, larger data packs are analyzed faster. Thankfully, this project was able to show us how to implement these algorithms and the troubles that may come with doing so incorrectly.

Screenshots:



Contributions:

Christian Lo: Helped contribute and formulate the puzzle function and contributed to implementing the A* algorithms. Also helped formulate the structure and the conceptual basis for the lab report.

Ethan Liu: Created the make puzzle function which allows users to create a new 8 puzzle or choose a default puzzle. Helped with heuristic for misplaced tile and euclidean distance. Also helped implementing Uniform Cost Search and A* algorithms.

Calvin Trujillo: Worked on getting some of the User Interface done and helped out other individuals within the group on their portions of the code. Also ran testing on parts of the code where issues were arising. Helped in fixing some of the issues that came out when running the program with certain parameters.

Krishaan Patel: Worked on the design and classes of the Project. Laid out the methods needed for the Node and Problem classes and then worked with the team to implement each of these methods (adding new ones as needed when formulating the search algorithms). Worked on creating the apply_operator and frontier checking methods