Alberto Gonzalez Trujillo

862127298

8 Puzzle Report

Challenges this Project

There are a lot of problems I faced with this project. There are too many problems to count. But just remember that the path to success is always under construction. I had difficulties understanding why building an object would be helpful. I asked Charles for help. He explained to me in three minutes what would have taken me hours to figure out. I think one of my biggest problems was not modularizing my code. In my original uniform cost, there was a lot of spaghetti code. There was potential, but it needed inspiration. After modularizing it made things a lot easier.

Code Design

Uniform Cost + Missing Tile + Euclidean

```
struct VectorHash
```

Hash function that I get help from to be able to use unordered sets

```
int find_sliding_piece(vector<int> v)
```

Function finds the 0, or the sliding piece

```
void find_possible_goals(vector<int> v, int space)
```

Tells the compiler what moves are legal. Almost like a recursive function. It goes into another function. This goes to push_nodes.

```
void push_nodes(vector<int> v, int space, string path)
```

Here the compiler knows what move to do. Path tells the compiler what path to take. So it shifts the value. For uniform cost it will push the nodes onto the priority queue no matter what as long as it hasn't been visited, but not for the other ones.

Missing Tile

```
void test_goal_states(vector<int> v)
```

I used a talented algorithm here to find the amount of misplaced tiles. And in the end I made a min_heap priority queue that would choose the nodes with the least amount of missing tiles.

Euclid Distance

```
float distance_to_goal(int space, int valid_move)
```

Returns the distance between the value and its goal state. Does this for every legal move. Kind of spaghetti implementation, but we take what we can. Same thing as missing tile. I have a min_heap as well. So the nodes that are prioritized are nodes with the smallest distance.

Code optimization

I decided to use unordered sets. There's linear time lookups. So let us say if I want to see if there already exists a node with the same exact puzzle, not to visit it again. So I used unordered sets to hold my visited nodes, because of how fast it is to check if the vector with the puzzle already exists inside. Uniform Cost is basically a brute force solution. Misplaced tiles prioritizes the tiles that are misplaced. Surprisingly very fast. For euclidean distance it solves the oh boy problem in 64 moves. Almost instantly. But for some reason it takes like 10000 moves for an even harder puzzle.

Search

I did tree searches, because I think graph searches, although they are cooler, would take a heck of a lot more time to solve. I was not able to keep track of the paths. I guess I'm just not that talented.

Findings

Uniform cost is slow. To Find the oh boy problem it takes around 180,000 nodes to be created. It also takes a lot of time to print the vector of the puzzle values. If I take out some of the print functions, even Uniform Cost finishes almost instantly. I think my heuristics could be better, but like Frank Ocean says, “We all try”. However, missing tile and euclidean distance are pretty fast. It solves the rest of the puzzles very fast. Like I said, I could maybe tweak it even more.