Adele Fuchs
P1 Extra Credit

For the extra credit portion of the project I chose to use my aStarSearch and beamSearch methods (and their helper methods) to solve a 15 puzzle.

Code Design:
I represented the 15 puzzle in the same way that I did my 8 puzzle, as a string. Most changes needed to convert the 8 puzzle to the 15 puzzle were a matter of changing limits on row and column lengths and board size. However, due to the fact that the 15 puzzle has tiles that contain multi-digit numbers, I had to alter the implementation of a few methods including: stateToBoard and the main method. In stateToBoard which converts a string of the state into an array of strings, I had to account for spaces in the state string which had been inserted to deal with differentiating multi digit numbers. In the main method which reads the input file, I had to alter the switch statement for setState to accommodate the fact that the board could now only be represented as a string of the following format “0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15”.

In the solver class, a few key changes were made for the same reasons as above, to change limits and account for the spaces needed in the state string. The method that was altered the most was the generateNextNodes method. This method divided a state’s string into rows and columns and then performed various operations to move around the 0. When spaces and multi digit numbers were introduced to the state string it became too difficult to implement this method in this way because the rows extracted would need to be of variable length. Rather than doing operations on and directly manipulating the state string, I decided to instead create a new puzzle here and use the move operation and toString method which were already defined in the puzzle class to generate the next possible states. This could also have been done in the 8 puzzle to simplify the method’s implementation.

Although change was needed in the helper method that calculates the possible next states of a given state, no changes were needed to the structures or direct implementation of either a* search or beam search.

Code Correctness: A lot of the code correctness overlaps with that of the original project and thus this version is less extensive
I have attached sample text files that show both the basic functionality of both classes as well as where the A* searches and beam search fail for the 15 puzzle.

Experiments: Not all questions addressed as some of them are repeats

I wanted to test what the cutoff was for the number of moves that the a* search could solve. To do this I looked up the 15 puzzle state that required the most moves to solve which was 15 14 8 12 10 11 9 13 2 6 5 1 3 7 4 0 and apparently requires 80 moves to solve (https://puzzling.stackexchange.com/questions/24265/what-is-the-superflip-on-15-puzzle). None

of the search algorithms were able to search this deep on my machine. The maxNodes limit was set to 100,000,000. And while none of the algorithms completely failed, they were not able to solve the problem in a reasonable amount of time. This shows that as the state space that the algorithms need to search increases(8 puzzle to 15 puzzle), the algorithm is less likely to be able to find a solution due to the number of nodes that need to be created and searched.

I decided to compare the number of nodes created by these algorithms when solving the 8 puzzle vs the 15 puzzle. I expect the number of nodes generated to be relatively similar for solutions of similar depths in the puzzles and that the searches will work up until roughly the same solution depth.

	8 puzzle results			15 puzzle results		
	Search Cost (nodes generated (N))			Search Cost (nodes generated (N))		
Depth (d)	A*(h1)	A*(h2)	Beam k=5	A*(h1)	A*(h2)	Beam k=5
1	4	4	4	4	4	4
2	8	8	21	7	7	16
3	11	11	58	9	9	43
4	16	13	313	26	16	1162
6	31	25	4589	32	19	4908
9	89	33	182044	99	190	372202
11	206	88	2668114	381	688	974579
12	206	50	7222837	728	216	Won’t Solve
14	569	324	Won’t solve	1109	355	Won’t Solve
15	1111	338	Won’t solve	6554	294	Won’t Solve
17	2358	347	Won’t solve	8265	7490	Won’t Solve
19	6240	697	Won’t solve	37447	18571	Won’t Solve
22	21093	2540	Won't solve	187751	78053	Won’t Solve
27	143594	10207	Won't	464928	227537	Won’t

			solve			Solve

Analysis of table results: A similar result to that obtained from the 8 puzzles is found. In general, both versions of the A* search still perform better than beam search. A surprising result for the beam search was that it was unable to compute 15 puzzles with solution depths equal to the depths of 8 puzzle solutions that were solvable. I am not sure why this would be as the algorithm and number of states stored in both cases were the same. Comparing the 2 heuristics for A* also displayed a similar result as for the 8 puzzles except for in a few outlier rows (9 and 11). Typically h2 performs better than h1 however, when acting on the 15 puzzle, there were a few cases where the number of misplaced tiles proved to be a better heuristic than the manhattan distances.

In general when comparing the number of nodes needed to solve a 15 puzzle vs an 8 puzzle of the same depths, more nodes are needed to solve the 15 puzzle than the 8. I believe this is because although in both puzzles from every state there are only ever 4 moves max, in the 8 puzzle, the rows and columns are shorter and thus more cases are “edges” sooner and only result in 2 or 3 moves rather than 4 (In the 15 puzzle there are more tiles that the 0 can move 4 directions from and these must be created as potential states even if their heuristic value is bad and they aren't part of the solution).

The nodeLimit has the same effect in the case of the 15 puzzle as it did in the 8 puzzle, the effect just occurs earlier since shorter puzzles require more modes to solve in the 15 puzzle than in the 8 puzzle. When a node limit is enforced, beam search’s abilities dwindle very quickly. The abilities of A* using h1 decrease quicker in general that those of A* using h2.

As was seen with the 8 puzzle, for all tested cases the A* searches found a solution with the same number of moves. Beam search also had the same number of moves for the puzzles which it could solve. I suspect that if beam search were able to solve some of the puzzles that had deeper solutions, it may come up with a different solution as it is not necessarily searching for the shortest path solution like A* does.

Since a 15 puzzle has more deeper solutions I believe that the fraction of solvable puzzles that my algorithms can solve for a 15 puzzle is far less than the fraction for my 8 puzzle implementation. A 15 puzzle can have a solution depth of up to 80 moves. Since my A* searches can solve up to around 30 moves deep, they can solve about 37.5% of puzzles. Since my beam search algorithm can only solve puzzles with up to around 11 moves, only about 13.75% of puzzles are solvable by my algorithm. (These are assuming there are an equal number of puzzles that require each number of moves to solve).