Matthew Daniel

CS 470

February 3, 2017

Sub Project 1a Write-Up

**Summary:**

I wrote a sliding puzzle program in the programming language “Swift.” I began by making the puzzle playable by a user via the command line. Next, I created a small application that allows a user to play multiple games, clicking on tiles to “slide” them around. I also added a feature that allows games to be played on 4x4 and 5x5 boards.

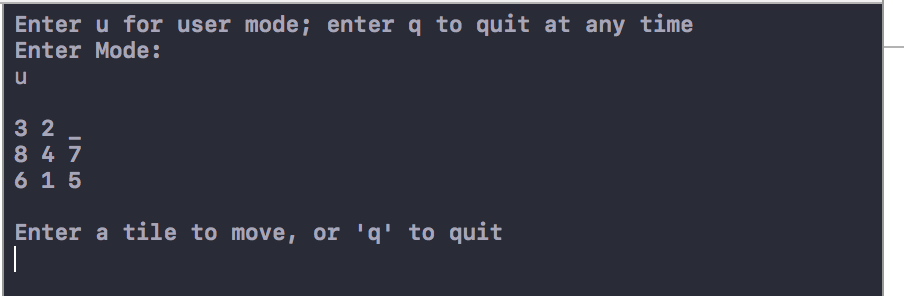


Figure 1: Beginning of a command line game

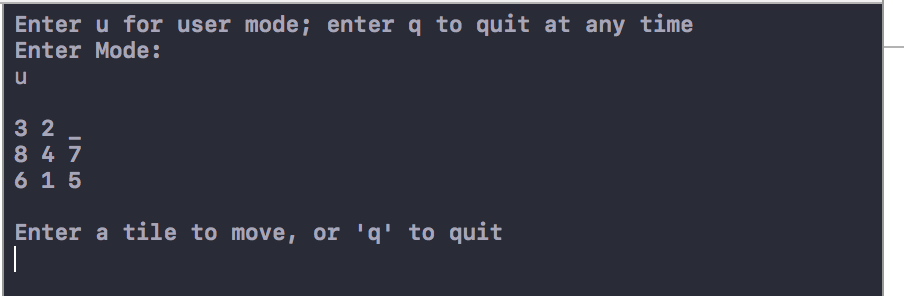


Figure 2: First move of a command line game

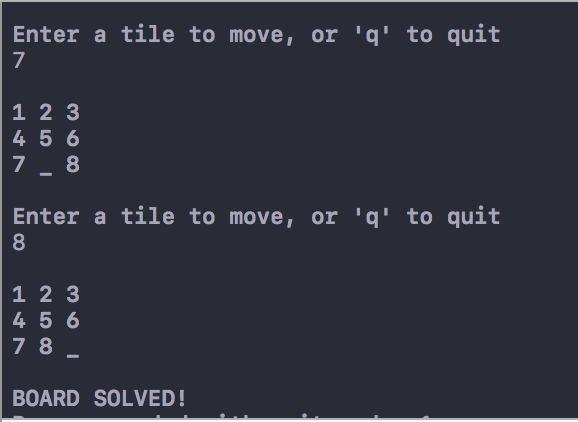
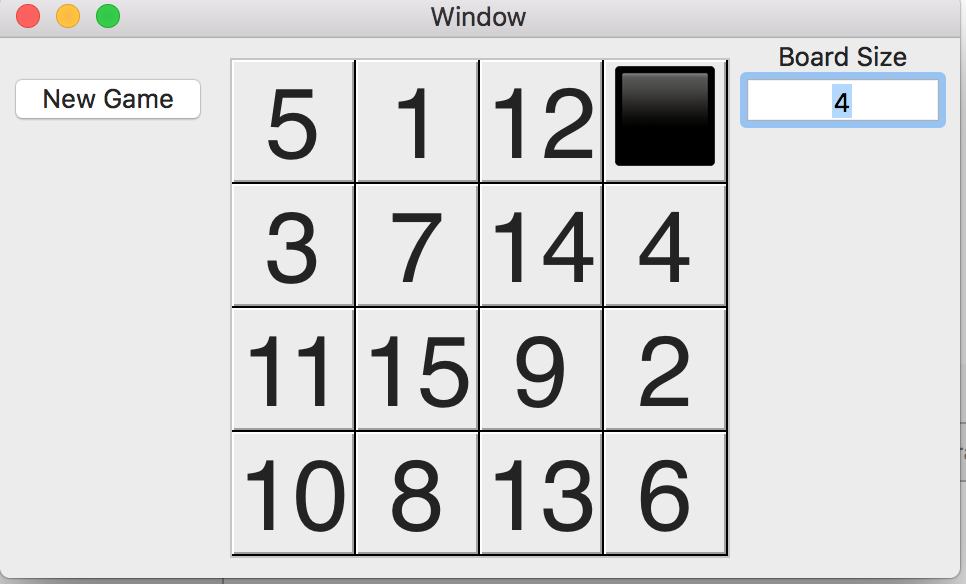
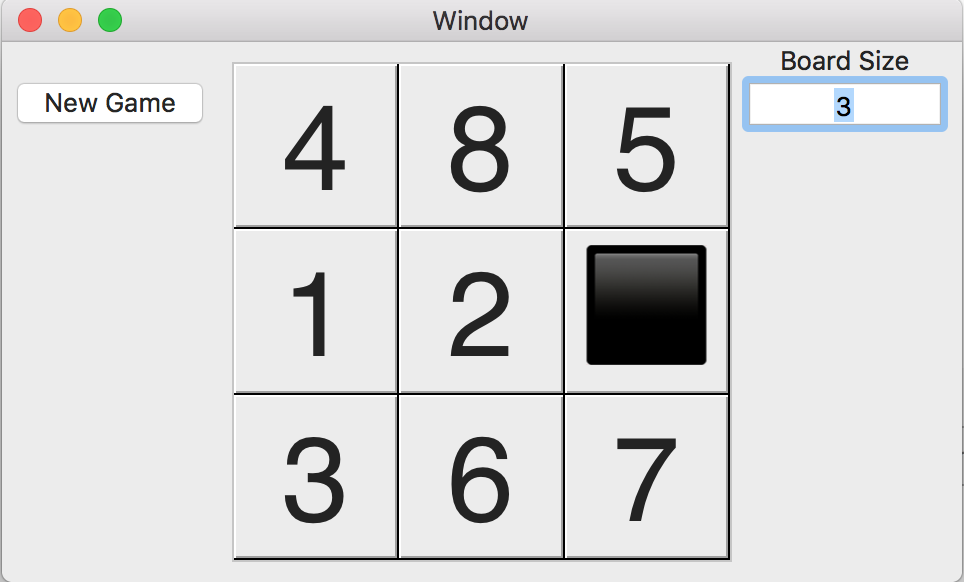


Figure 3: Final two moves



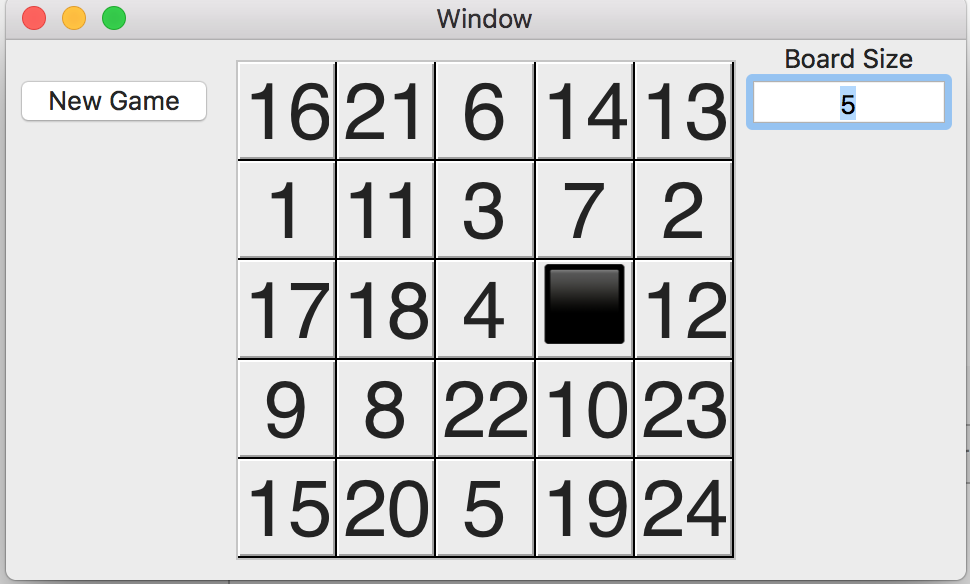


Figure 4: 3x3, 4x4, 5x5 Initial boards

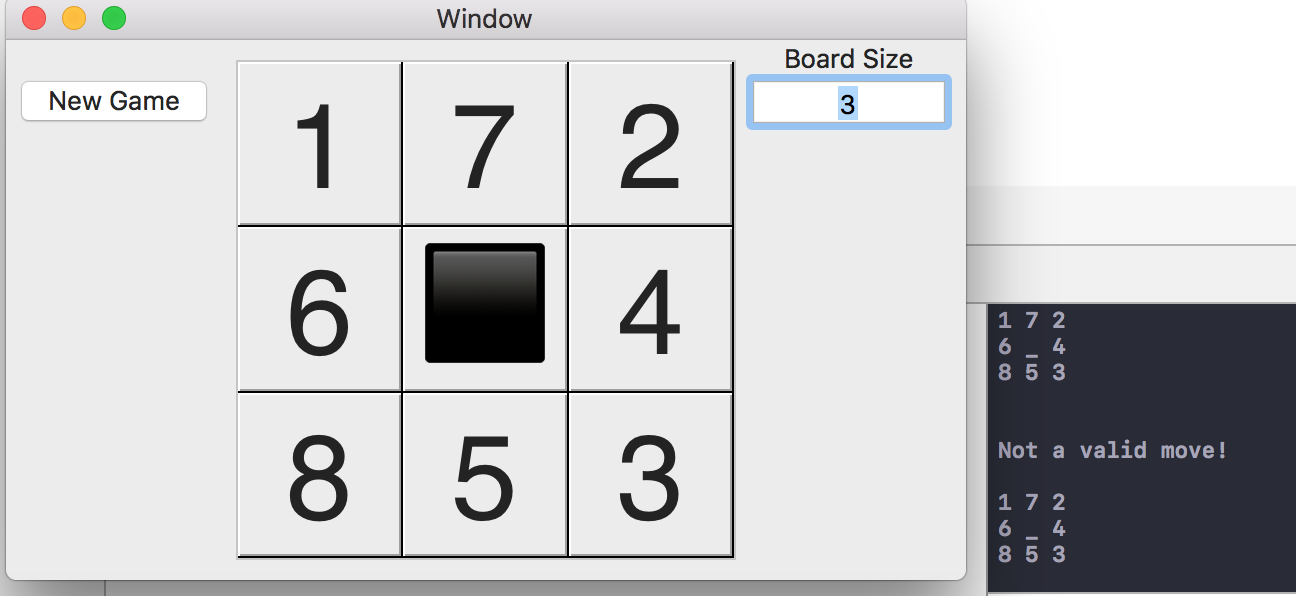


Figure 5: Gameplay. Note the simultaneous command line output

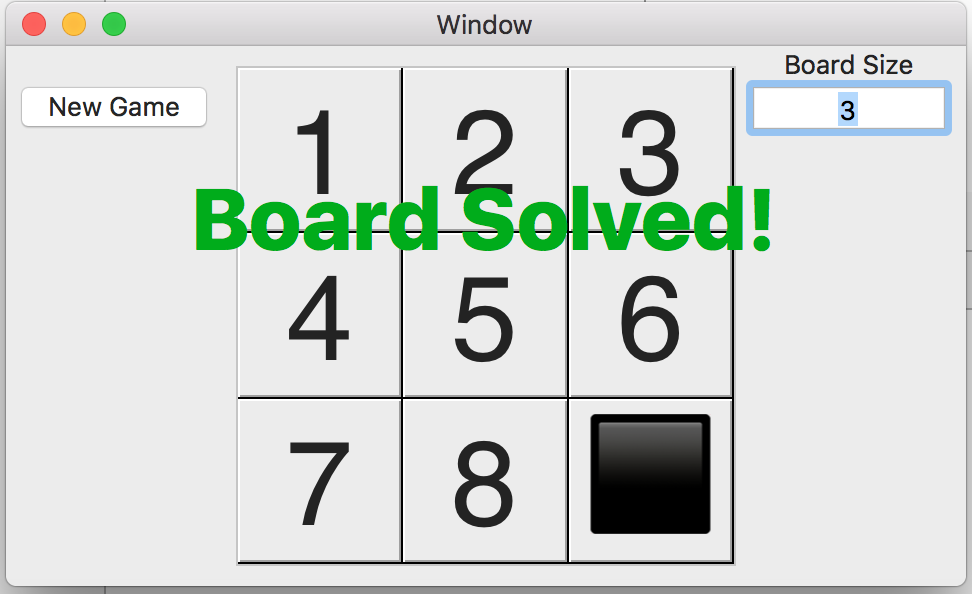


Figure 6: Solved board

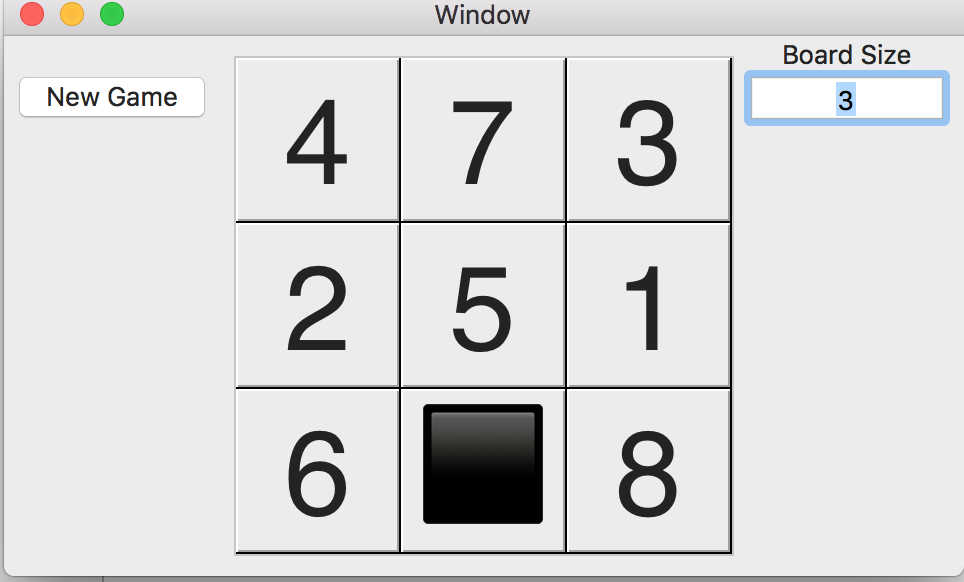


Figure 7: Start of New Game

Matthew Daniel

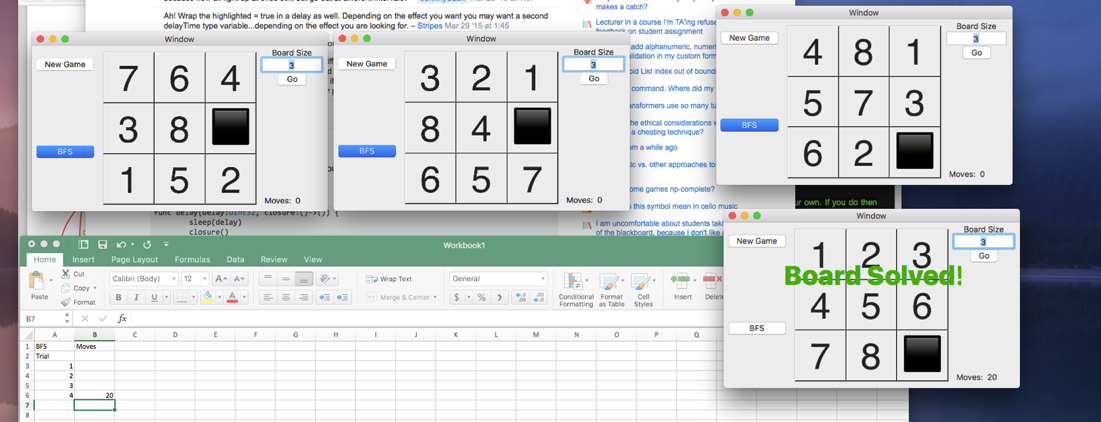
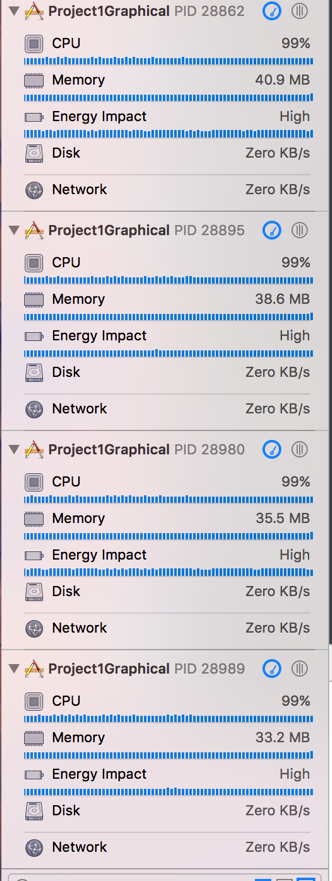
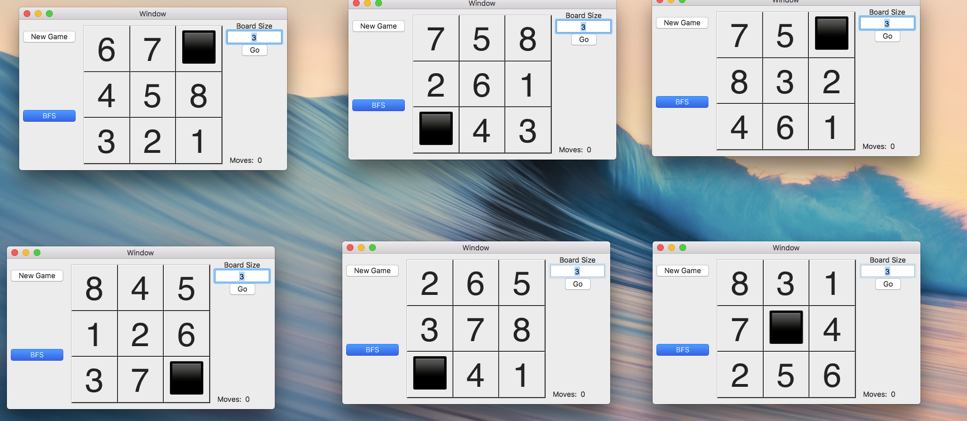
CS 470

February 10, 2017

Sub Project 1b Write-Up

**Summary:**

The breadth-first search solution to the sliding 8-puzzle found a solution in 20.77 moves on average, searching through an average of 72364 boards. Full results can be found on the attached spread sheet. I used a closed list, and did not check the moves as I put them on the open list. The sequence of moves used to solve the board by the BFS prints to the console, as well as is shown to the player via an animation (though the animation sometimes skips displaying moves). The function I used to represent boards in the closed list only works for solving 3x3 boards at the moment, though I may extend it to work for the 4x4 and 5x5 boards next week.



Screenshots of the application running and boards being solved.

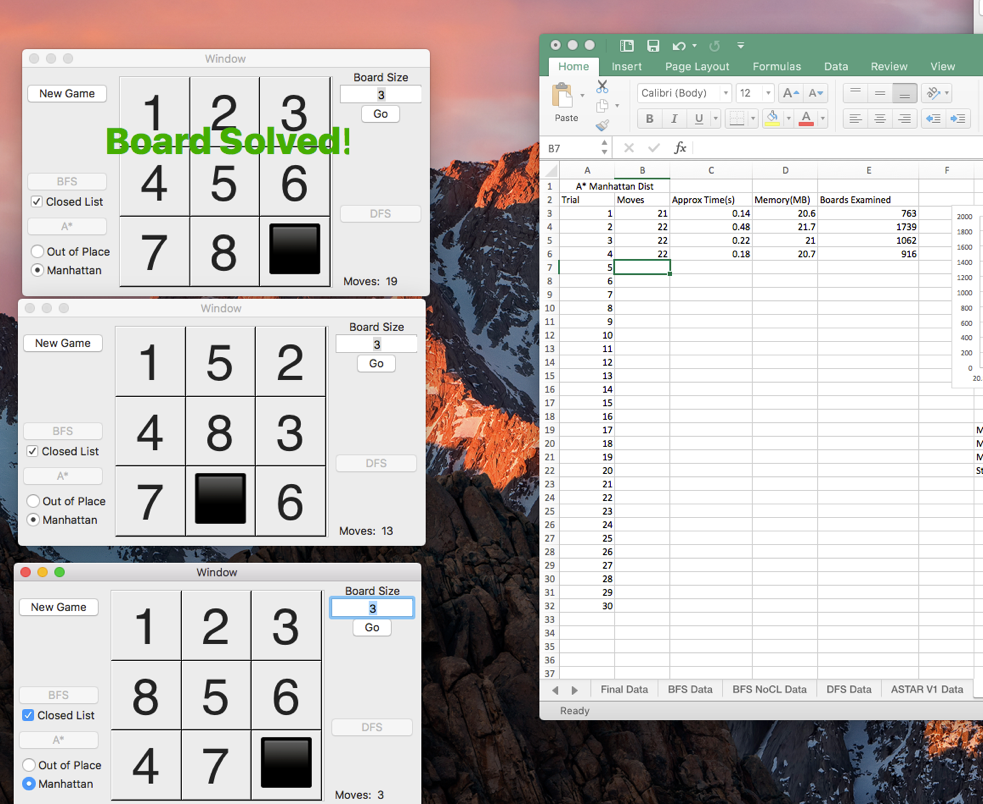
Matthew Daniel

CS 470

February 17, 2017

Final Project 1 Write-Up

**Summary:**  
I implemented the A\* algorithm with heuristic one being the number of tiles out of place and heuristic two being the sum of the distances of the tiles from their solved locations (Manhattan distances). These algorithms accompany BFS with and without a closed list and BFS. I ran each search 30 times, and the results of the number of moves taken, nodes searched, memory used, and time taken are attached. All the results are for the 3x3 board, as the 4x4 board did not run the algorithms within their space/time limits. I believe that this could have been addressed had I written a priority queue with faster insertion. Boards are randomized for 1000 times the board size moves.



Screenshot of three boards being solved with A\* search using the Manhattan distance

**Addition:**

I added the iterative deepening algorithm to the project after completion of the others. Its solution is nearly the same as the BFS, but it only adds nodes to the open list below the current depth limit, and goes to the next depth limit only if the open list is empty. For some reason, my iterative deepening search is not always optimal.