Logan Kelsch

COSC 411- Fall 2024

Project 2 Report

INTRODUCTION

In this project, we were asked to take our 15-Tile-Game code and implement a bot to solve the puzzle. This bot is supposed to be built with an A* searching algorithm. We were asked to clearly define a heuristic function and then demonstrate the bots performance.

HEURISTIC FUNCTION

The heuristic function I decided on during this project was formulated in the logic I take into account while solving the puzzle, and the limitations of a 1 depth search algorithm. In this program, I did not limit the algorithm to 1 depth observation fringe, as this was simply much too limited of a range to be capable of calculating this. I also considered forming a full tree before executing moves, but decided against this.

My custom heuristic function is mathematically based on manhattan distances of specific tiles. The function is broken into chunks to be more digestible as a single value to the algorithm. The splits in this function were labeled ‘chunks’, as it appears the bot solved the puzzle in the order of these chunks.

DEFINITION:

- Total of 5 chunks.
- The chunks were tile values of 1-2, 3-4, 5-6, 7-8, 9-13.
- For each chunk that remained unsolved, an additional value of 20 points of incorrectness was added to the board score.
- This means, as each chunk is solved, 20 points are taken off of the h() function.
- Within each chunk, an additional manhattan distance value is added for the chunk tiles.
- Specific additional punishing values were added to boards with inverted pieces to avoid long strings of moves required for fixing.

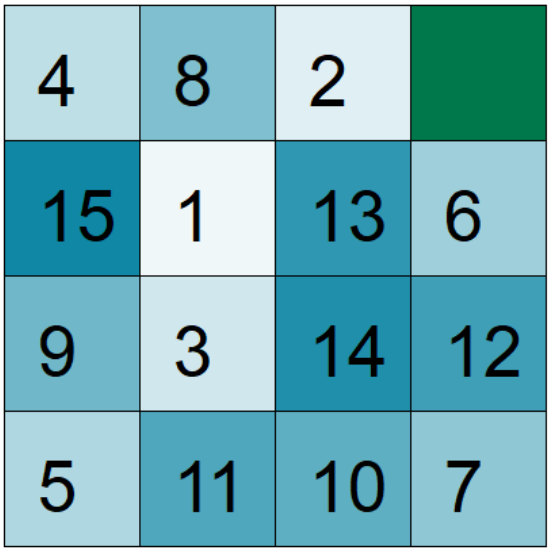
INTERPRETATION:

The use of these chunks listed in the definition of the heuristic function makes the current distance from a perfect board more digestible for the algorithm, while the punishing values allow the algorithm to avoid pitfalls of backwards tile positions.

MODEL PERFORMANCE:

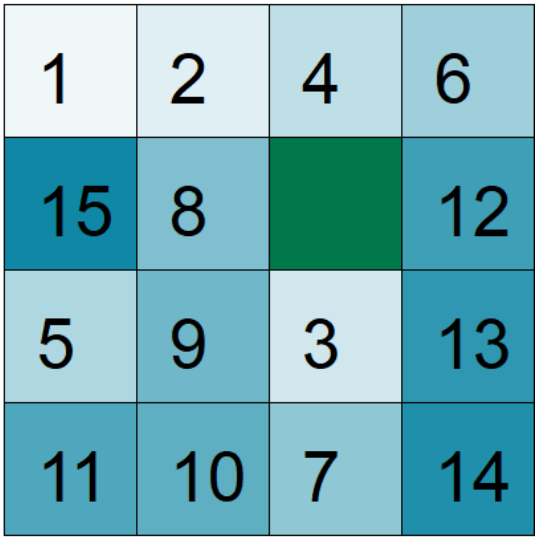
For this section, I have taken a screenshot of the bot completing each chunk during a single puzzle solve.

INITIAL CONFIGURATION



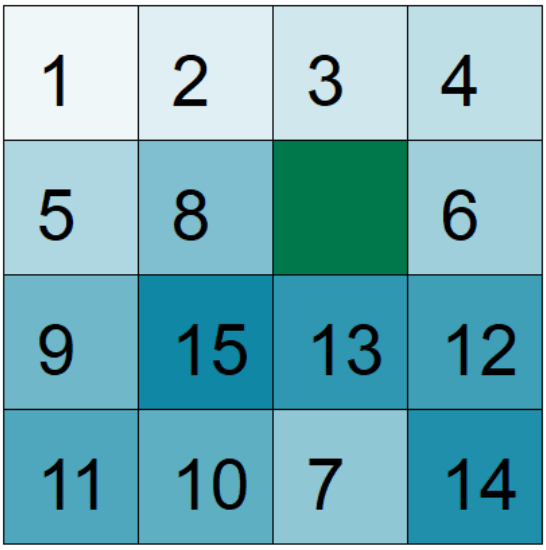
MOVES: 0

FIRST CHUNK COMPLETION



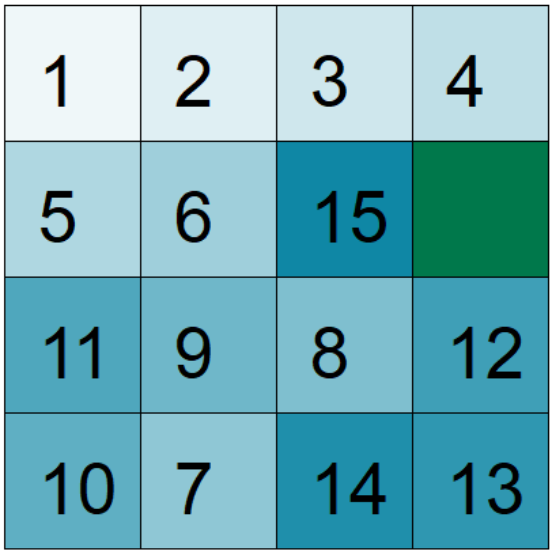
MOVES: 23

SECOND CHUNK COMPLETION



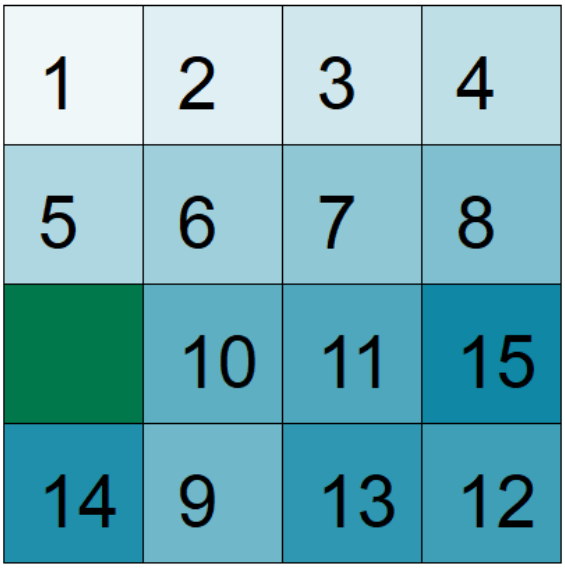
MOVES: 37

THIRD CHUNK COMPLETION



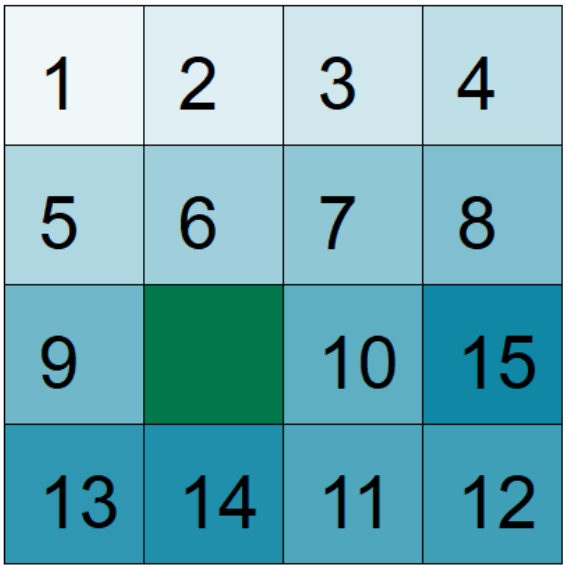
MOVES: 54

FOURTH CHUNK COMPLETION



MOVES: 77

FIFTH CHUNK COMPLETION



MOVES: 91

FINAL CONFIGUATION

This was also the best performance I had seen from my algorithm!

This algorithm averages around 200 moves for completion.

SOURCES CONSULTING

During this project, I did not refer to any sources or reach for any help except for the use of QTimer and its implementation which allowed me to easily implement a looping function call within the class and class initiation. I did also briefly discuss the difficulty of building this bot with JJ in the class, although we had no intention of working together and discovered we were using entirely different approaches.