Artificial Intelligence: Local Search

Nathan Morgenstern, Seo Bo Shim

September 26, 2017

Contents

Introduction to the Graphical User Interface(GUI)

When the Graphical User Interface starts up the user is able to select the type of puzzle evaluation through a drop down menu. The given options include: Basic Puzzle Evaluation, User Generated Puzzle Evaluation, Basic Hill Climbing, Hill Climbing with Random Restarts, Hill Climbing with Random Walk, Simulated Annealing, and Population Based Approach.

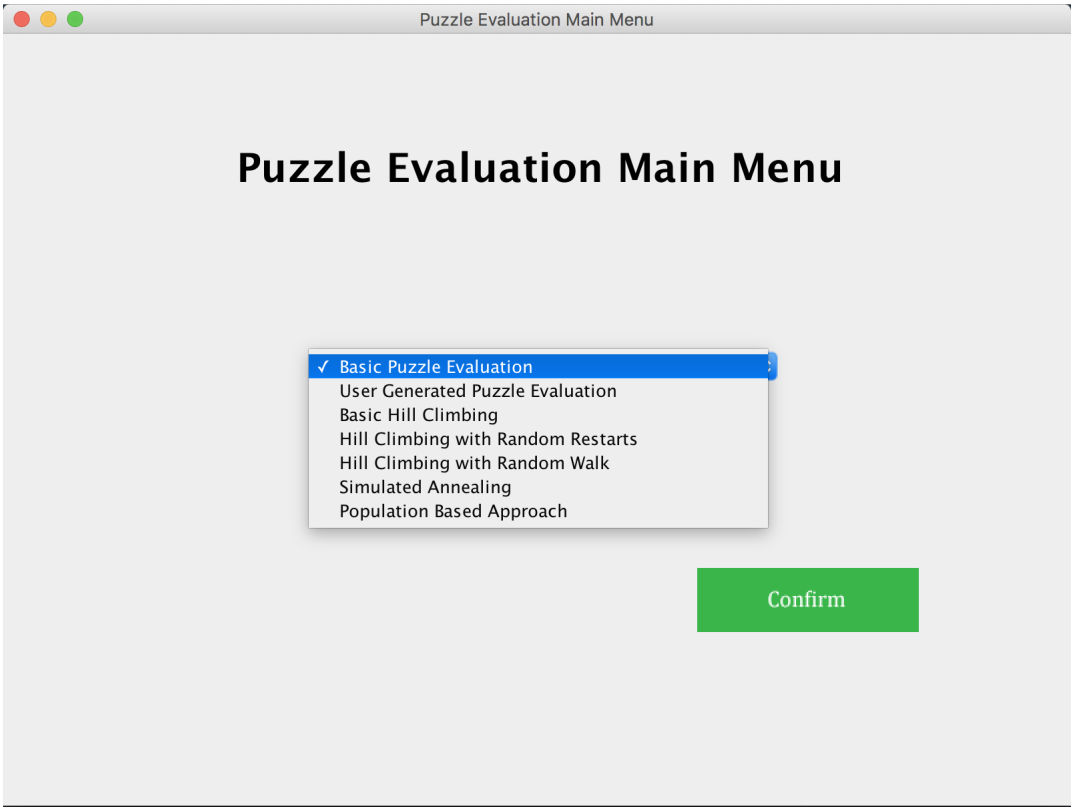

Figure 1: Main Menu of GUI

Each of the options then have their own corresponding window which is comprised of four main tabs: Puzzle Initialization, Puzzle, Puzzle Moves, and Data. The Puzzle Initialization tab is slightly different for each option in regards to the type of input received. The tabs Puzzle and Puzzle moves provide the user with a graphical representation of the generated puzzle as well as a graphical representation of the number of moves that it takes to get to each cell respectively.

Basic Puzzle Evaluation

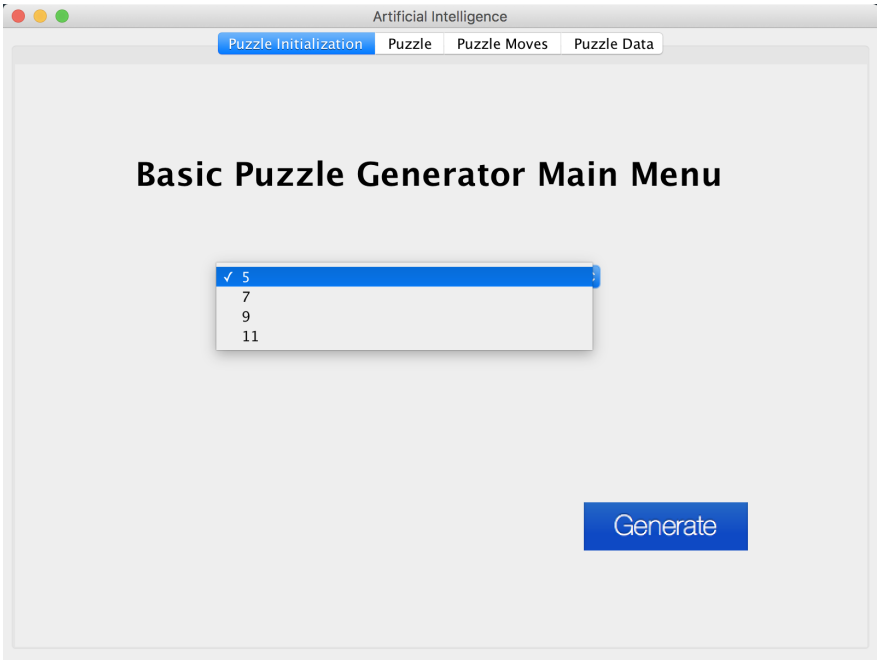

Figure 2: Puzzle Initialization of Basic Evaluation

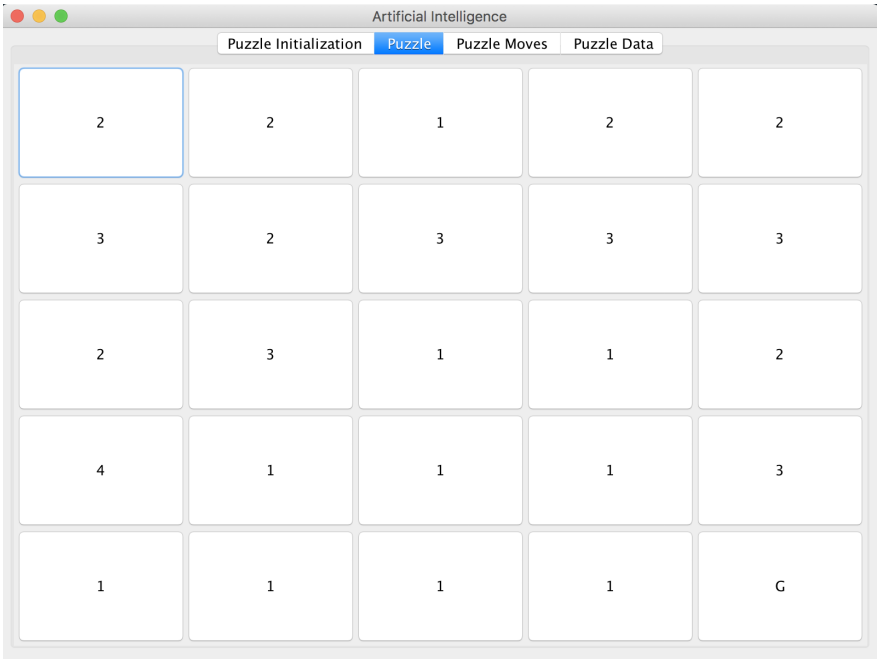

Figure 3: Puzzle Tab of Basic Evaluation

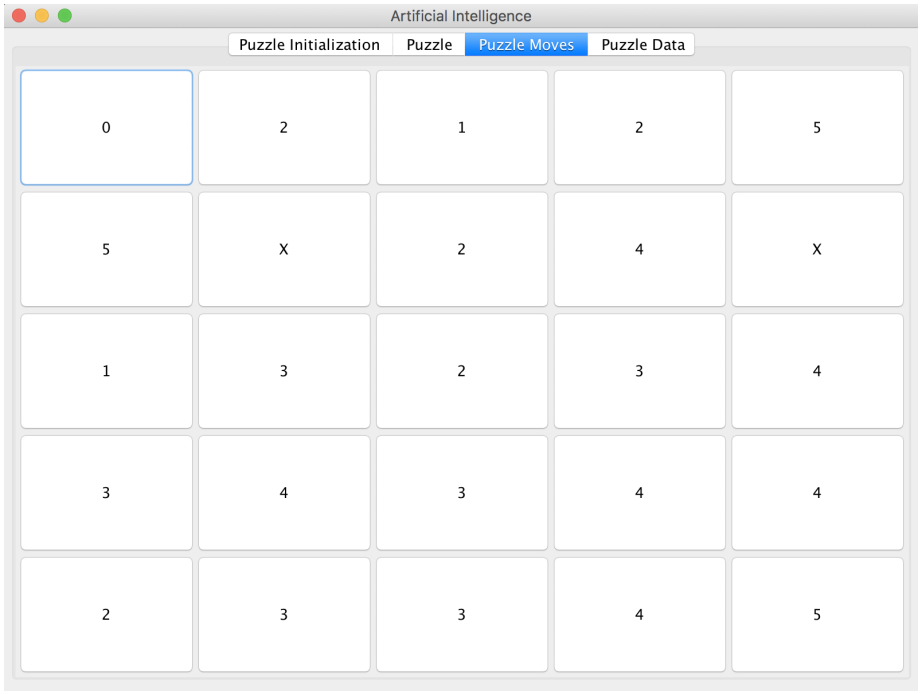

Figure 4: Puzzle Moves Tab of Basic Evaluation

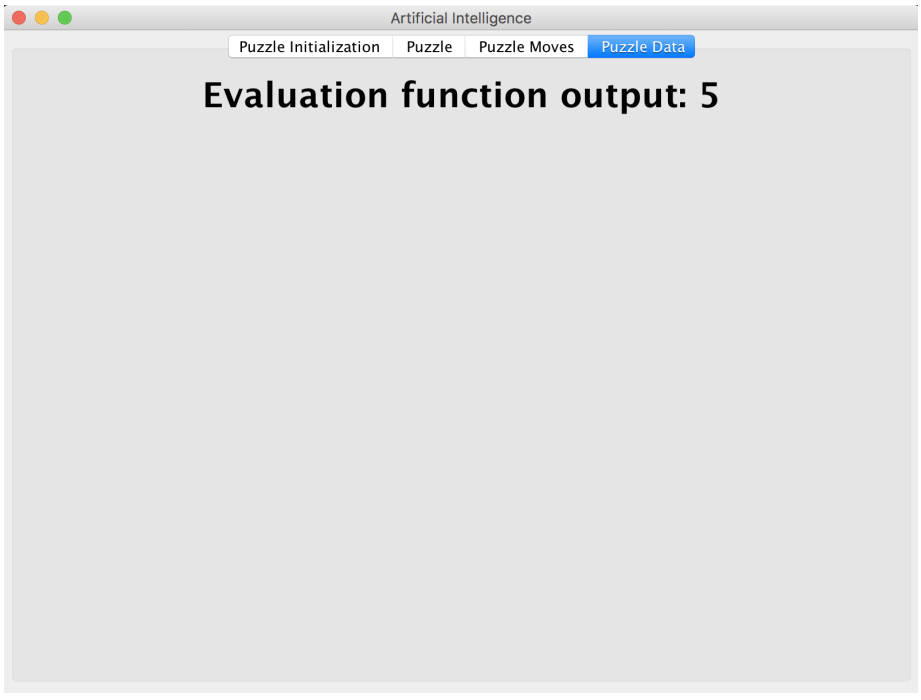

Figure 5: Puzzle Data Tab of Basic Evaluation

User Generated Puzzle Evaluation

The User Generated Puzzle Menu starts up with the default file of ./userPuzzles/assignment.txt, the user is able to change the file name to any location that they wish. The tabs Puzzle, Puzzle Moves, and Puzzle Evaluation are the same as the above examples for the Basic Puzzle Menu.

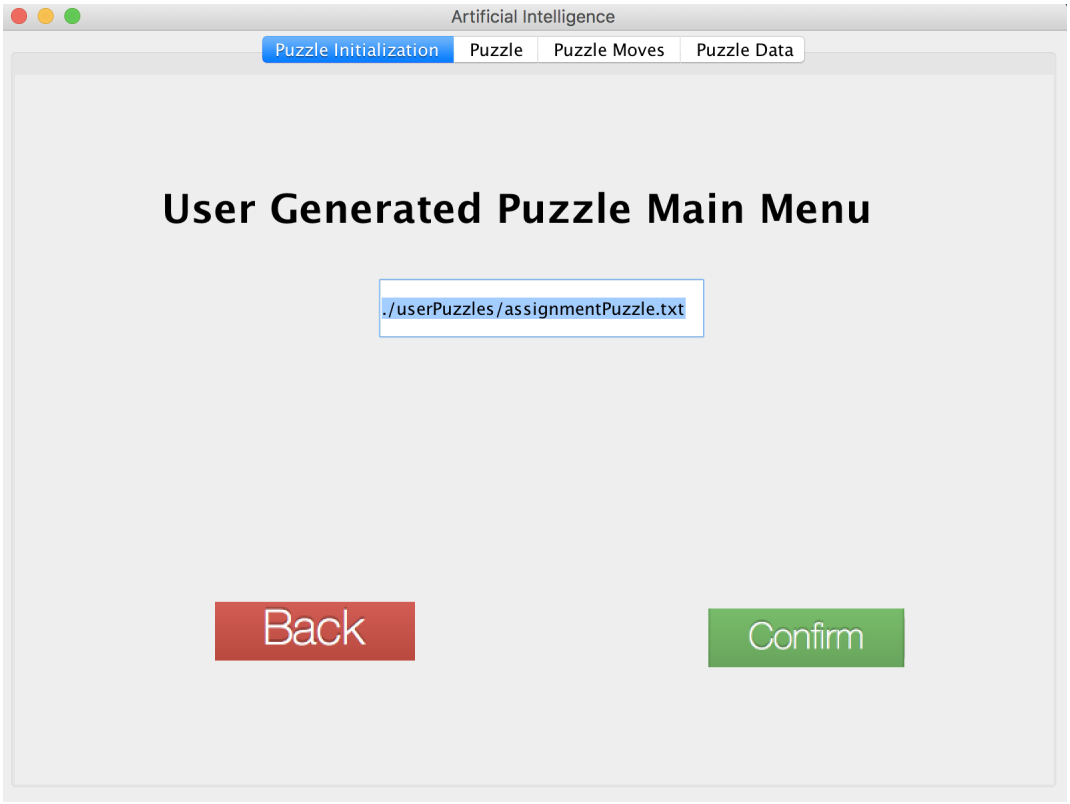

Figure 6: Puzzle Initialization Menu for User Generated Puzzle

Basic Hill Climbing

The Basic Hill Climbing Menu allows input for the size of the puzzle as well as a number of total iterations to perform the hill climbing algorithm. The data menu differs from the first two options in that it now shows a total evaluation time in addition to the evaluation function output.

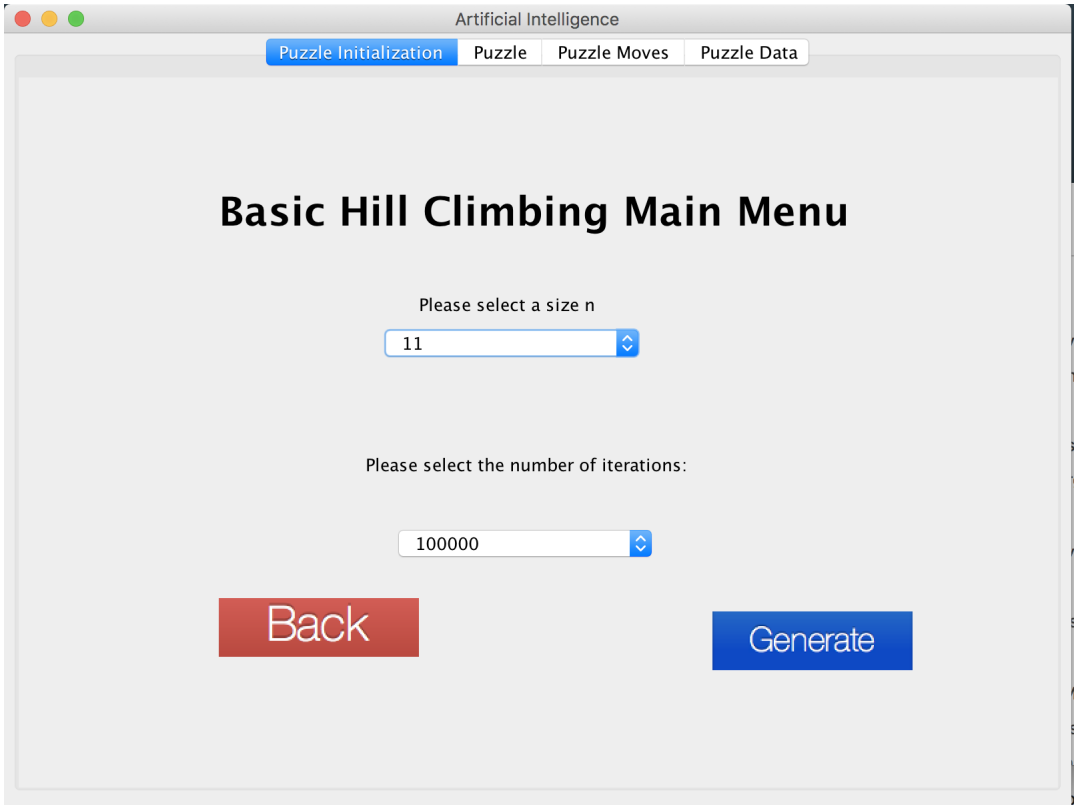

Figure 7: Puzzle Initialization Menu for Basic Hill Climbing

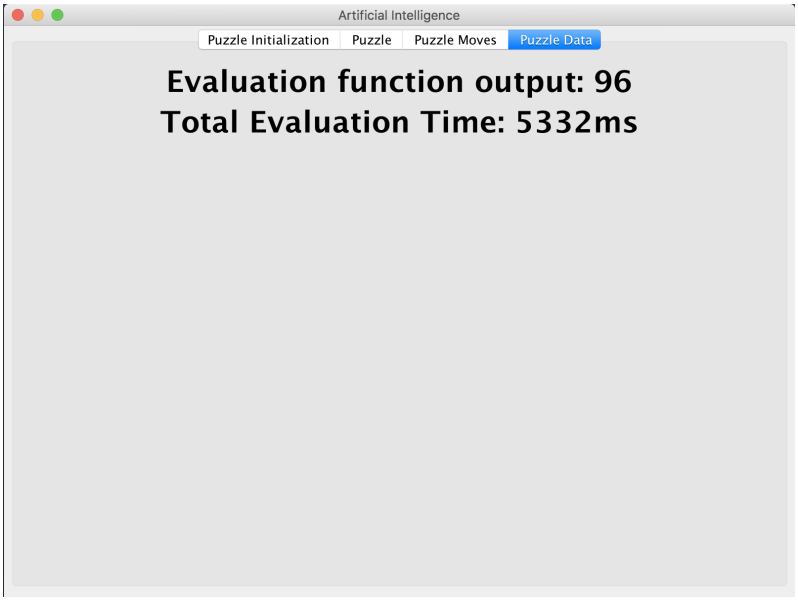

Figure 8: Puzzle Evaluation Tab for Basic Hill Climbing

Hill Climbing with Random Restarts

The Hill Climbing with Random Restarts Menu is similiar to the Basic Hill Climbing menu, with the addition of an input option for the number of random restarts that should be performed. The Puzzle, Puzzle Moves, and Evaluation tabs remain unchanged from the previous example.

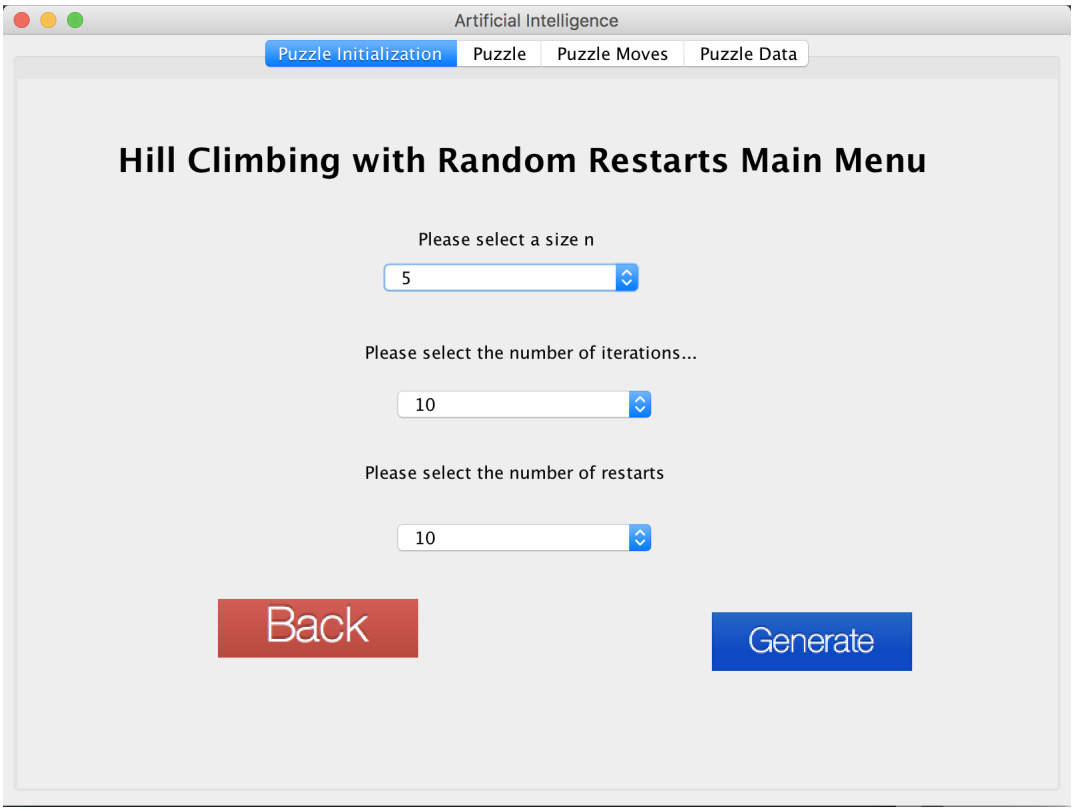

Figure 9: Puzzle Initialization Menu for Hill Climbing with Random Restarts

Hill Climbing with Random Walk

Hill Climbing with Random Walks Menu is the same as Hill Climbing with Random Restarts, but replaces the bottom input option of number of restarts, with the probability of down hill acceptance.

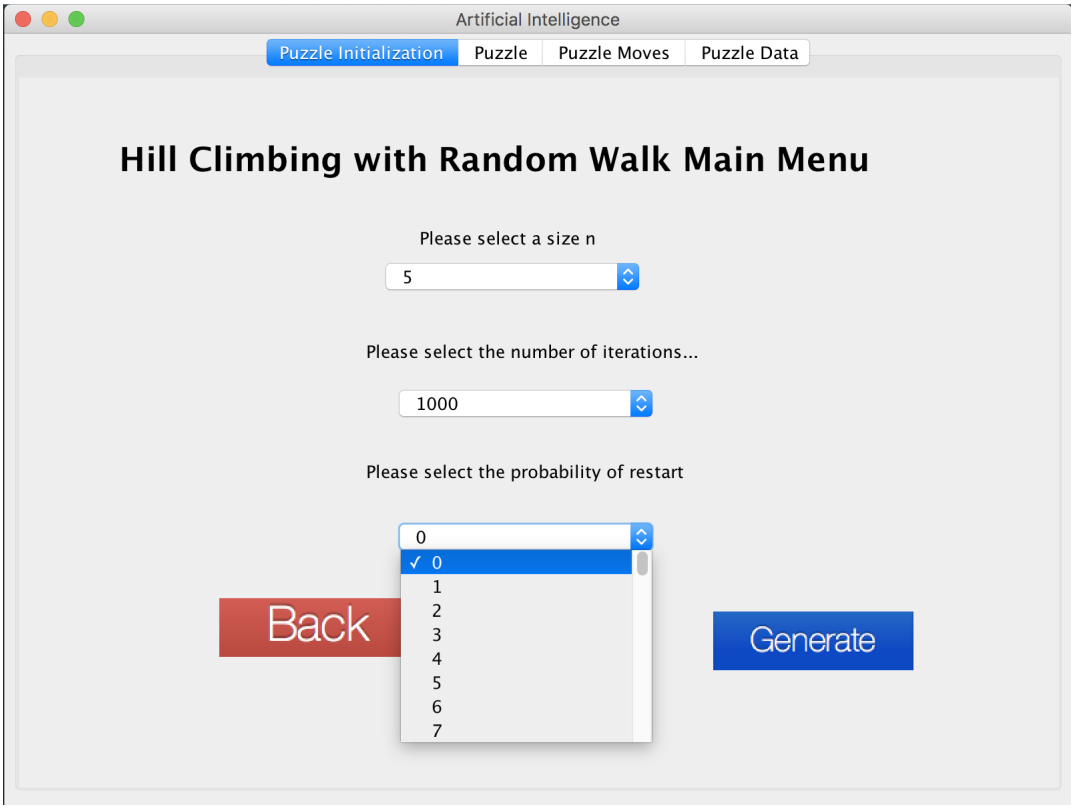

Figure 10: Puzzle Initialization Menu for Hill Climbing with Random Walk

Simulated Annealing

The initialization menu for Simulated annealing includes 4 input options: the size n of the puzzle of the puzzle, the initial temperature T, the total number of iterations, and the decay rate.

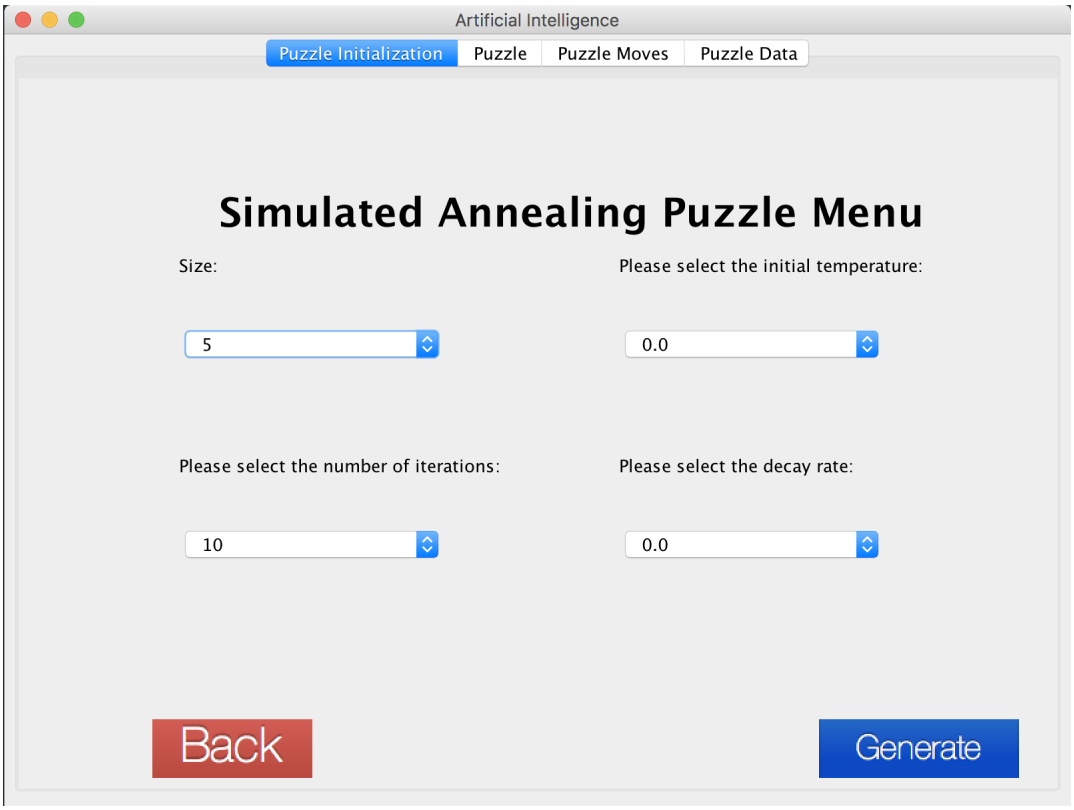

Figure 11: Puzzle Initialization Menu for Simulated Annealing

Population Based Approach

The population based approach menu includes input options for puzzle size, initial population size, total time to run the algorithm, and the probability for mutation.

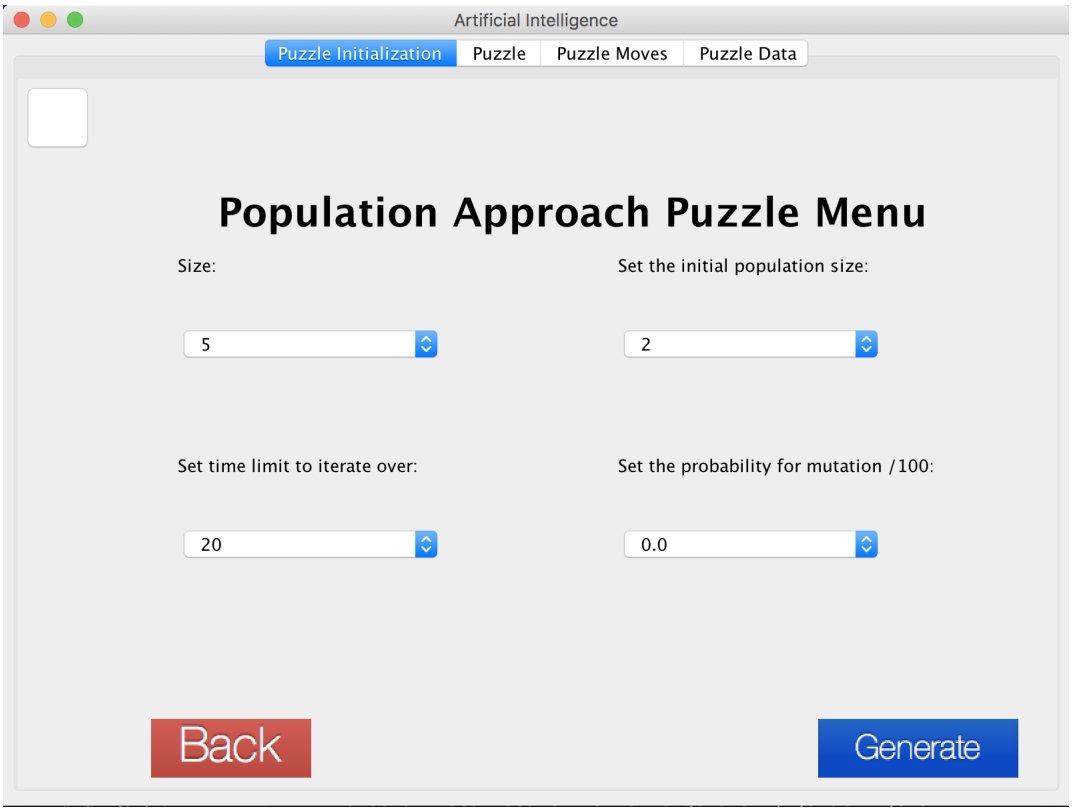

Figure 12: Puzzle Initialization Menu for Population Approach

Puzzle Evaluation

Example solvable and unsolvable puzzles for each size of n are included below. These puzzles were generated by the basic puzzle generator menu briefly discussed in the GUI section.

Example Puzzle for n = 5

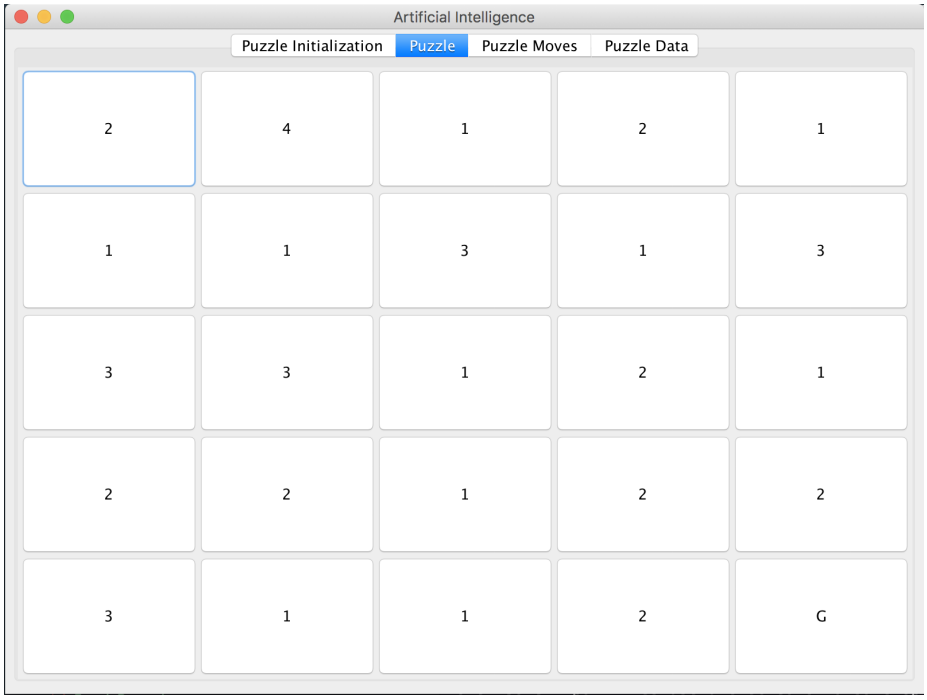

Figure 13: Reachable Goal Puzzle size n = 5

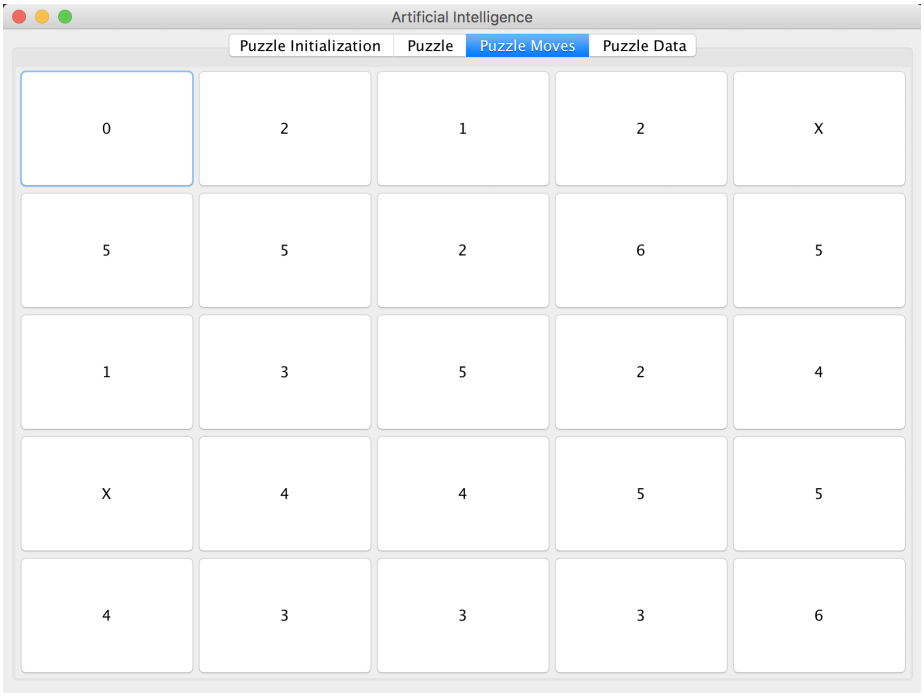

Figure 14: Reachable Goal Puzzle Moves size n = 5

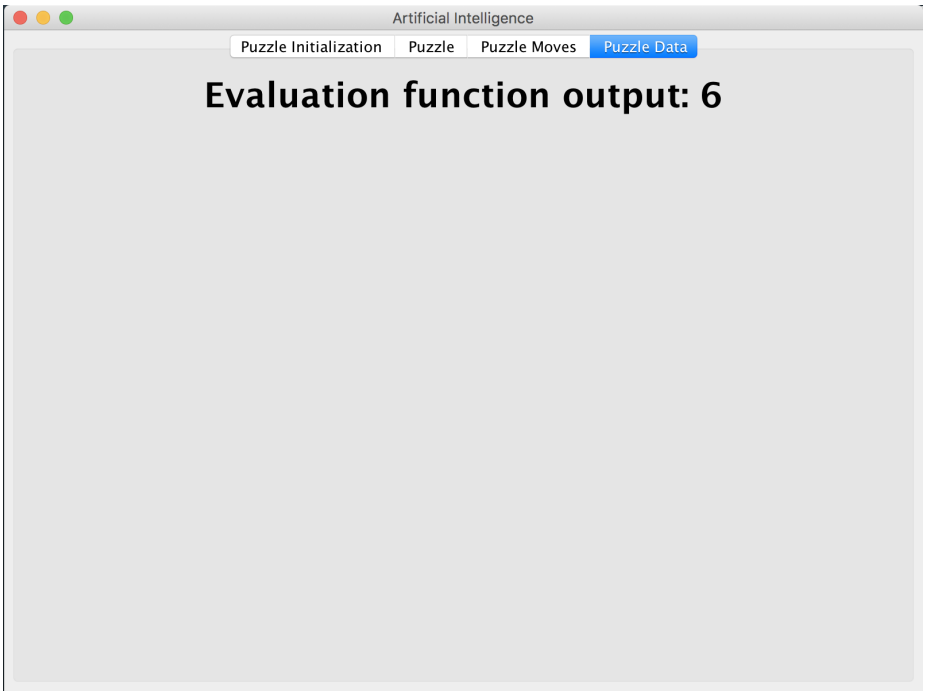

Figure 15: Reachable Goal Puzzle Evaluation size n = 5

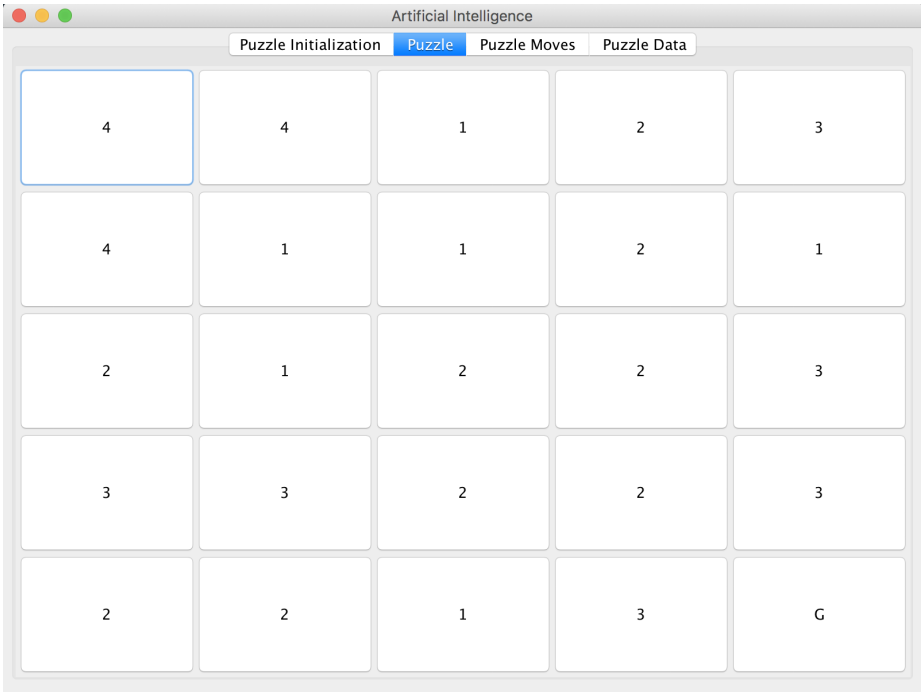

Figure 16: Unreachable Goal Puzzle size n = 5

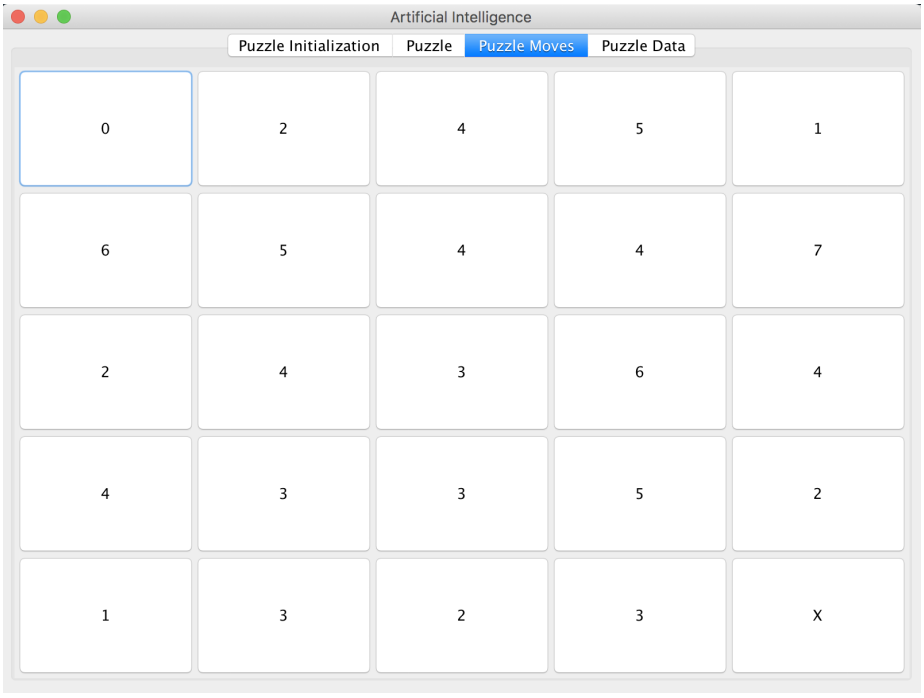

Figure 17: Unreachable Goal Puzzle Moves size n = 5

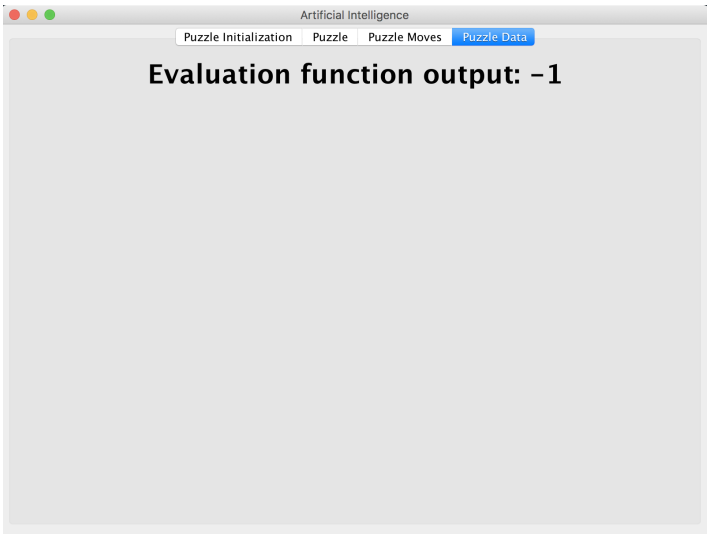

Figure 18: Unreachable Goal Puzzle Evaluation size n = 5

Example Puzzle for n = 7

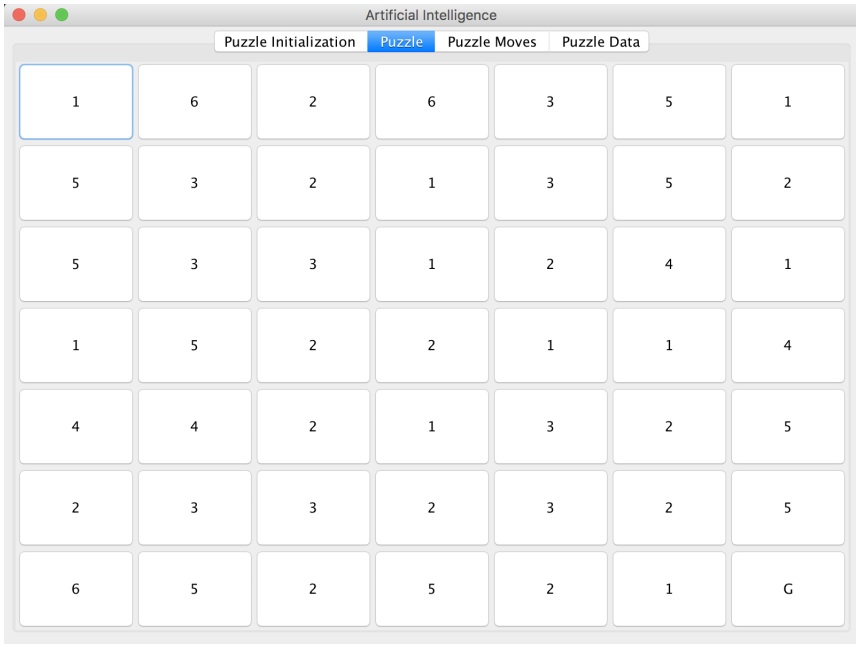

Figure 19: Reachable Goal Puzzle size n = 7

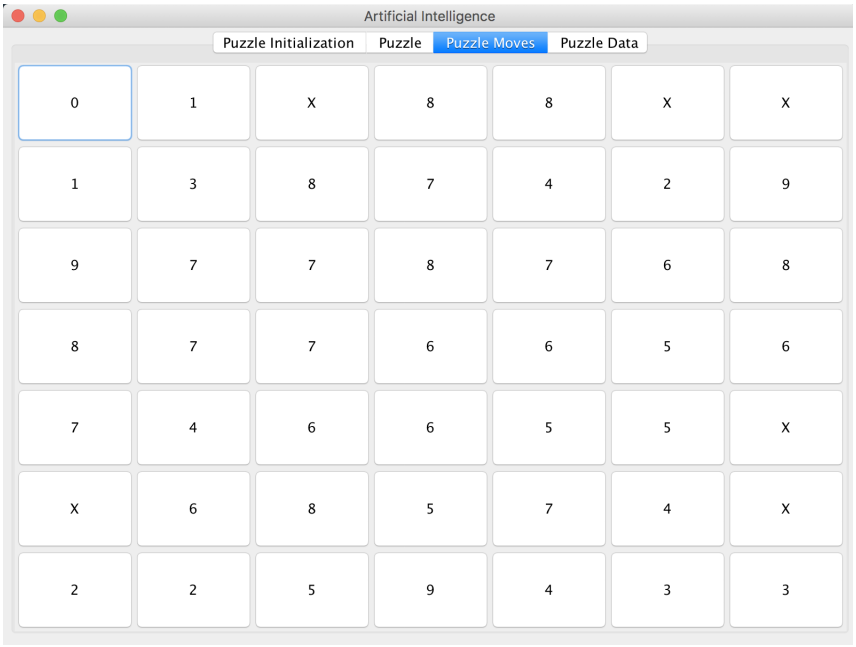

Figure 20: Reachable Goal Puzzle Moves size n = 7

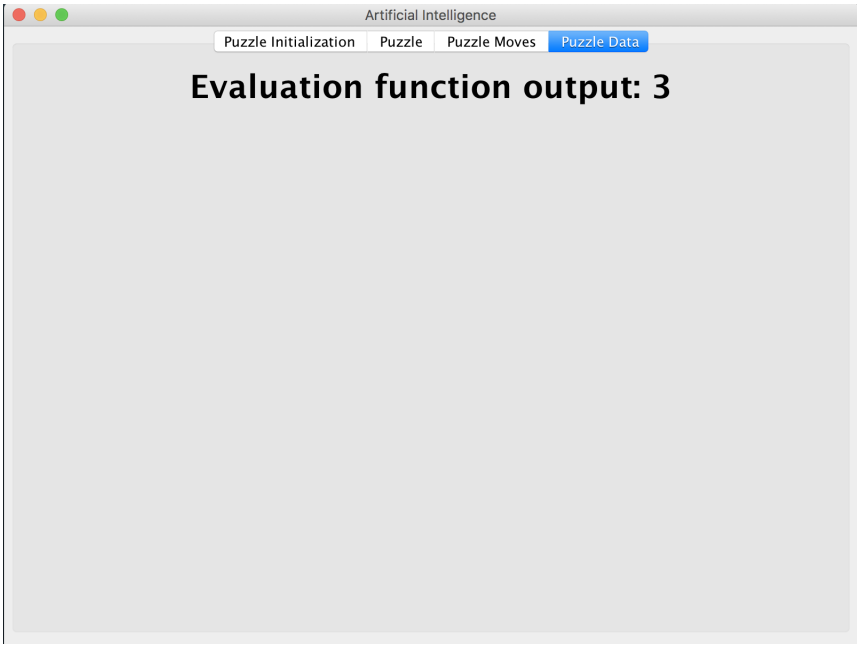

Figure 21: Reachable Goal Puzzle Evaluation size n = 7

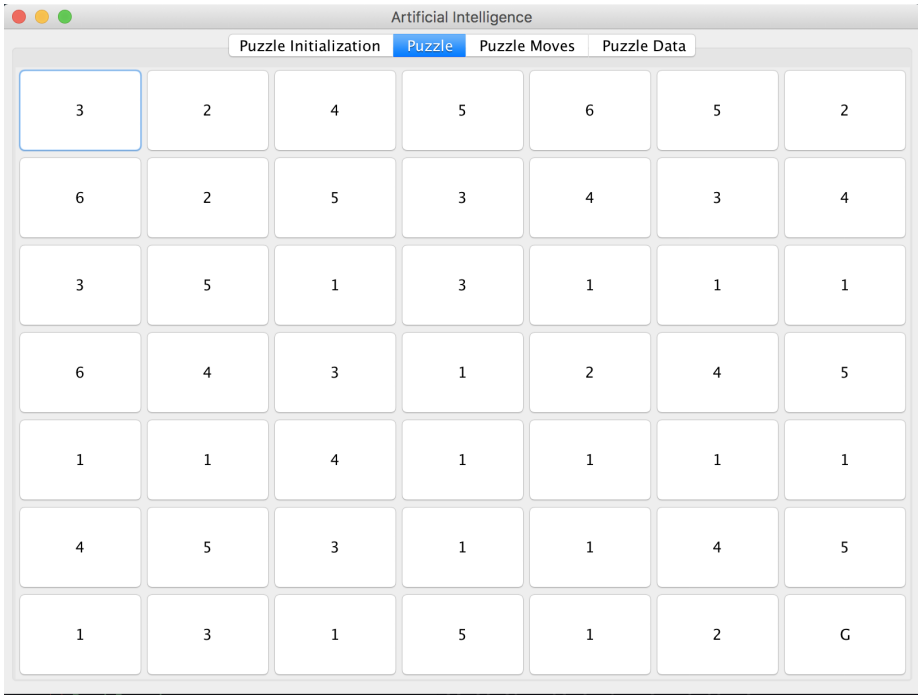

Figure 22: Unreachable Goal Puzzle size n = 7

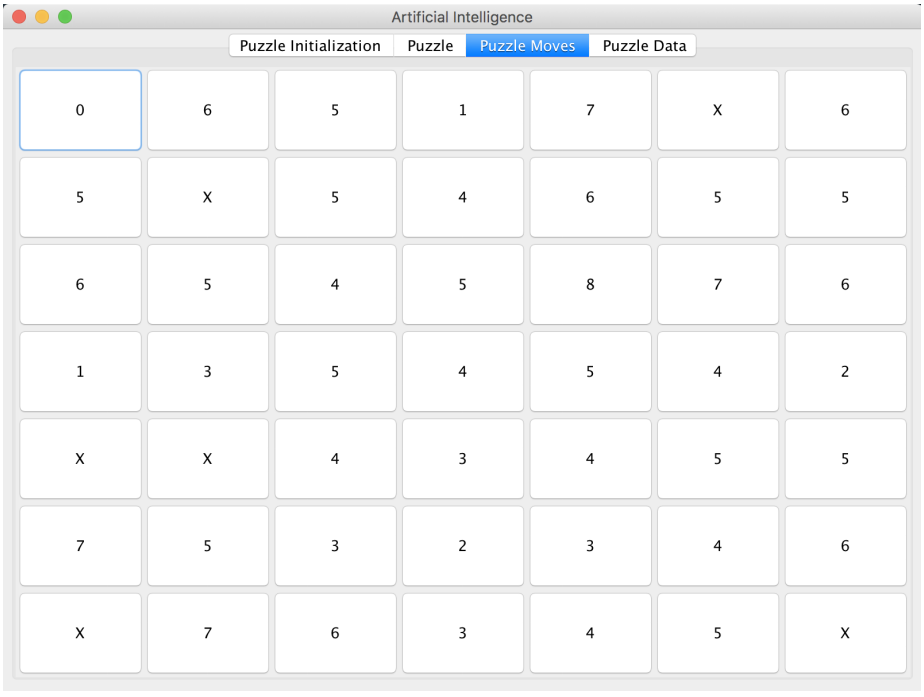

Figure 23: Unreachable Goal Puzzle Moves size n = 7

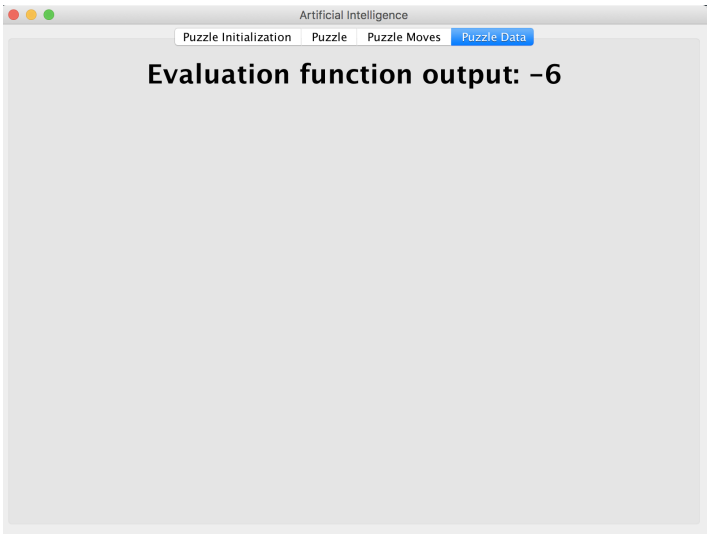

Figure 24: Unreachable Goal Puzzle Evaluation size n = 7

Example Puzzle for n = 9

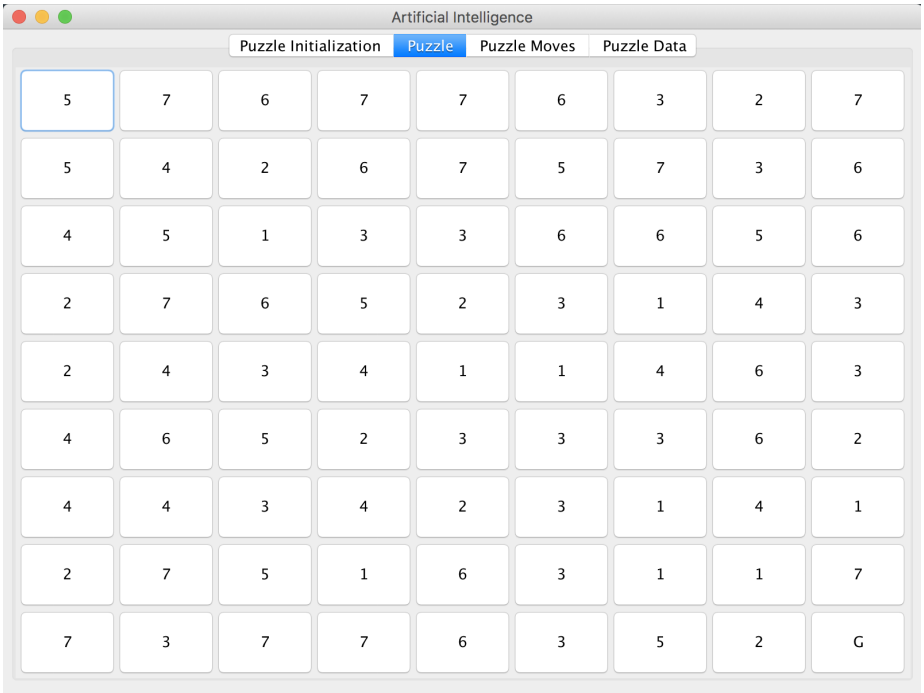

Figure 25: Reachable Goal Puzzle size n = 9

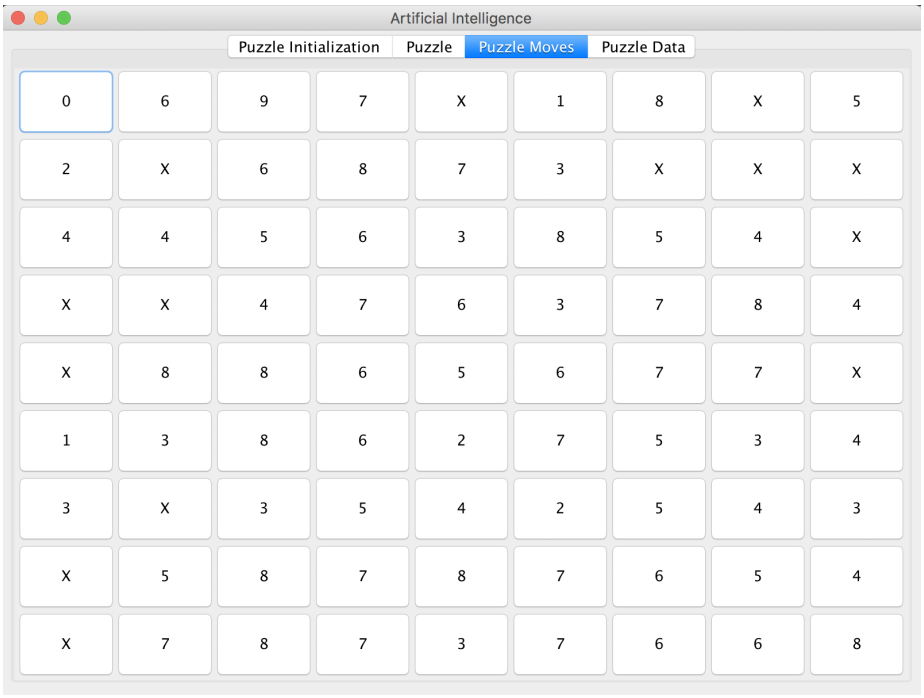

Figure 26: Reachable Goal Puzzle Moves size n = 9

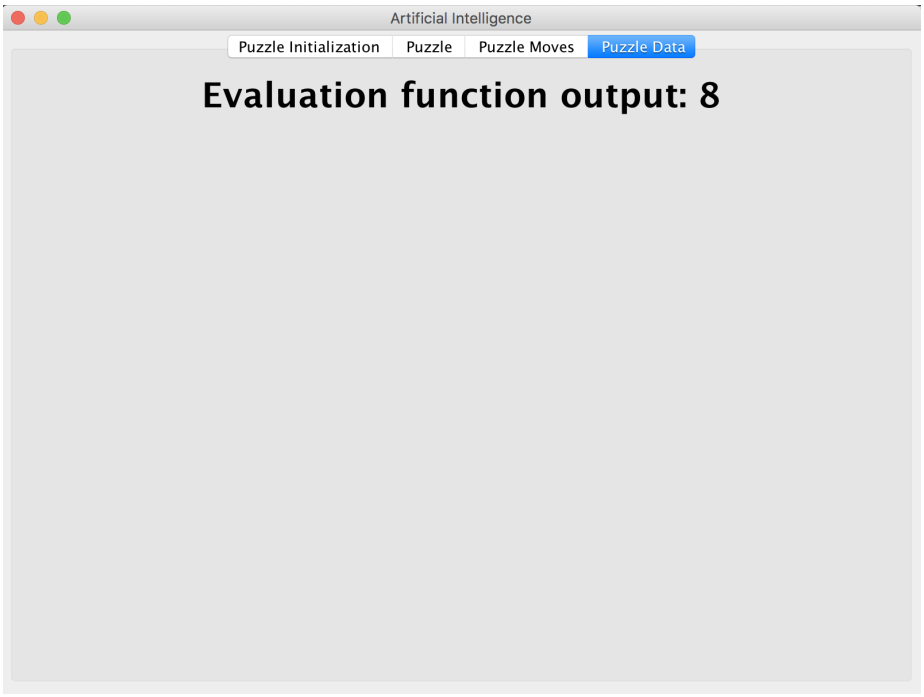

Figure 27: Reachable Goal Puzzle Evaluation size n = 9

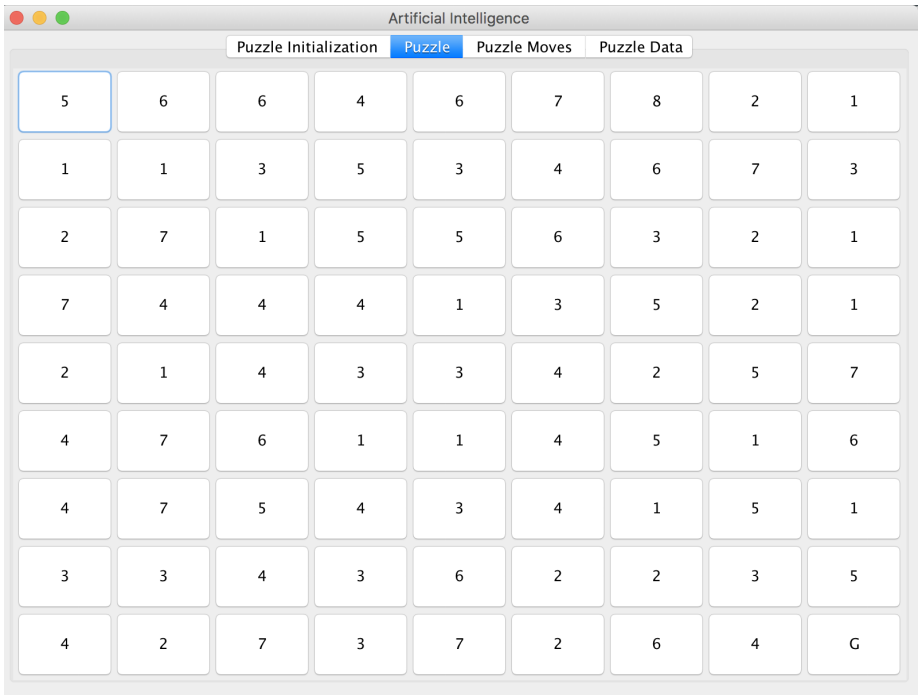

Figure 28: Unreachable Goal Puzzle size n = 9

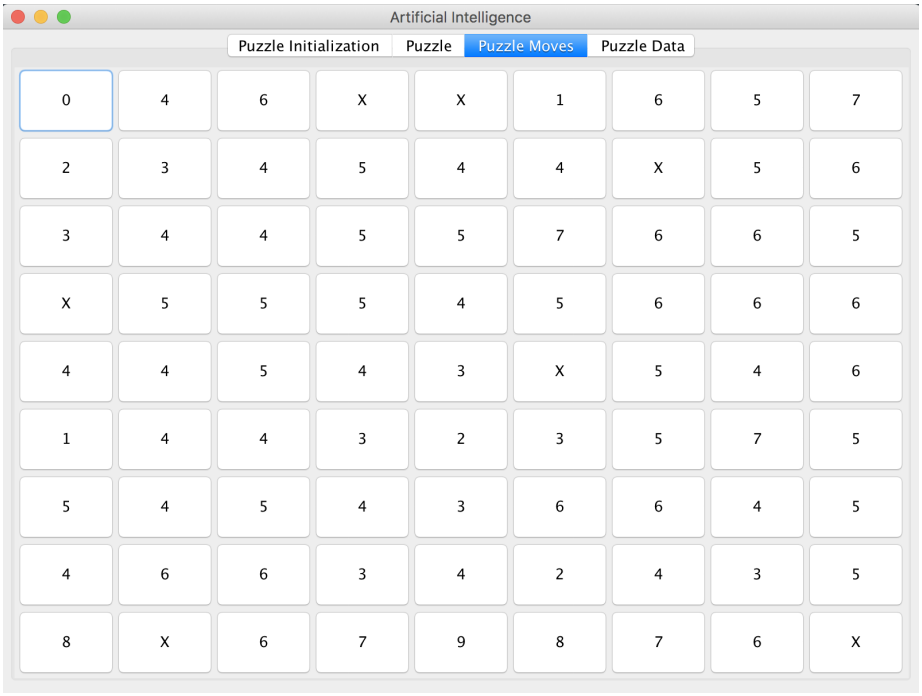

Figure 29: Unreachable Goal Puzzle Moves size n = 9

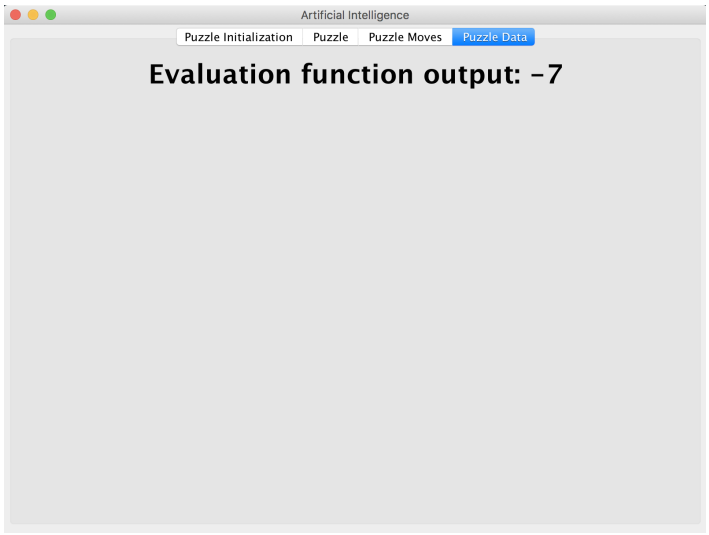

Figure 30: Unreachable Goal Puzzle Evaluation size n = 9

Example Puzzle for n = 11

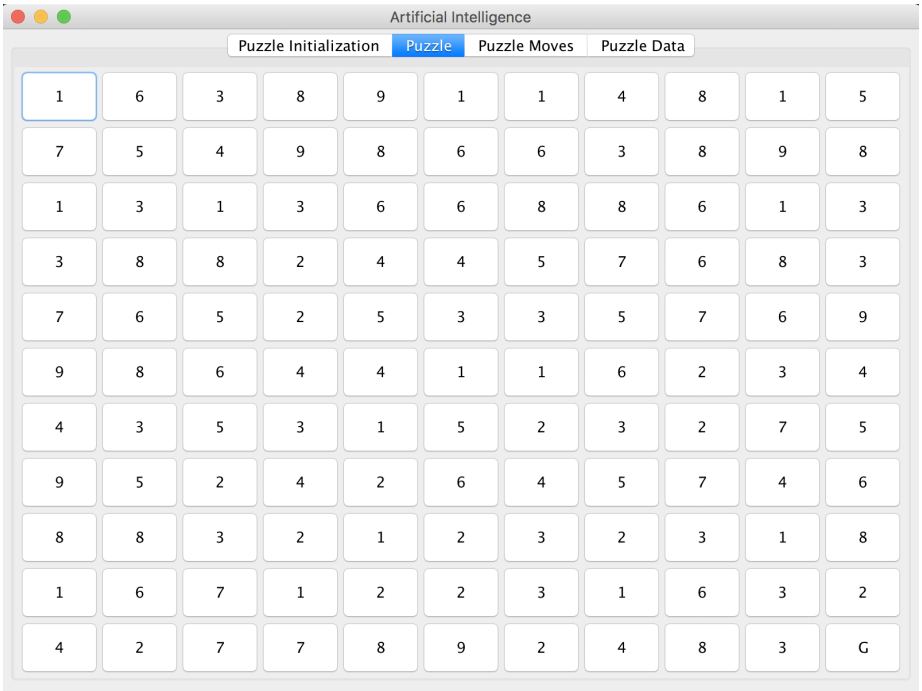

Figure 31: Reachable Goal Puzzle size n = 11

Figure 32: Reachable Goal Puzzle Moves size n = 11

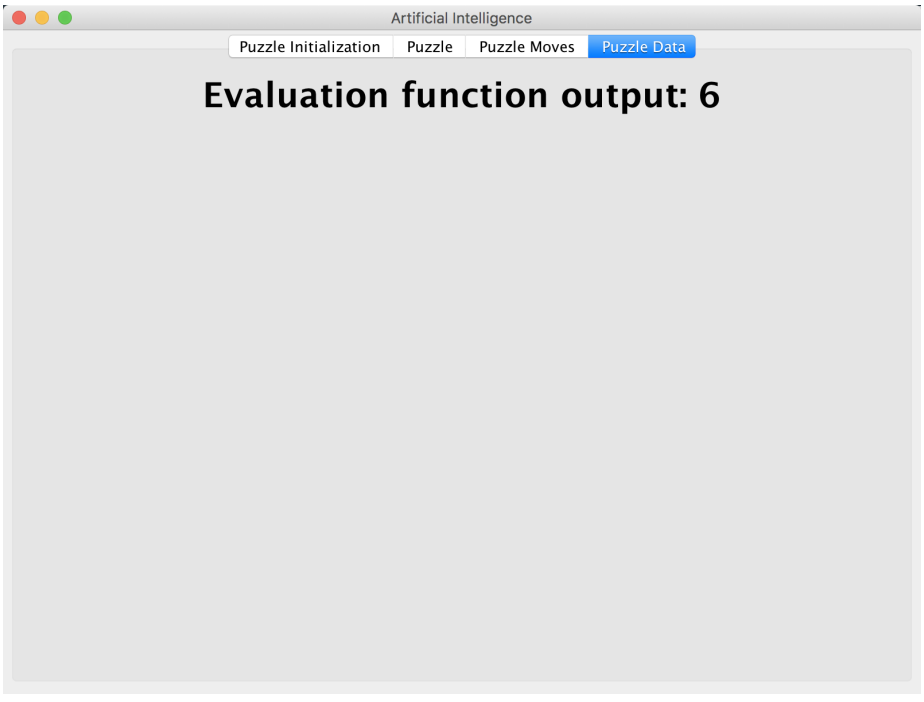

Figure 33: Reachable Goal Puzzle Evaluation size n = 11

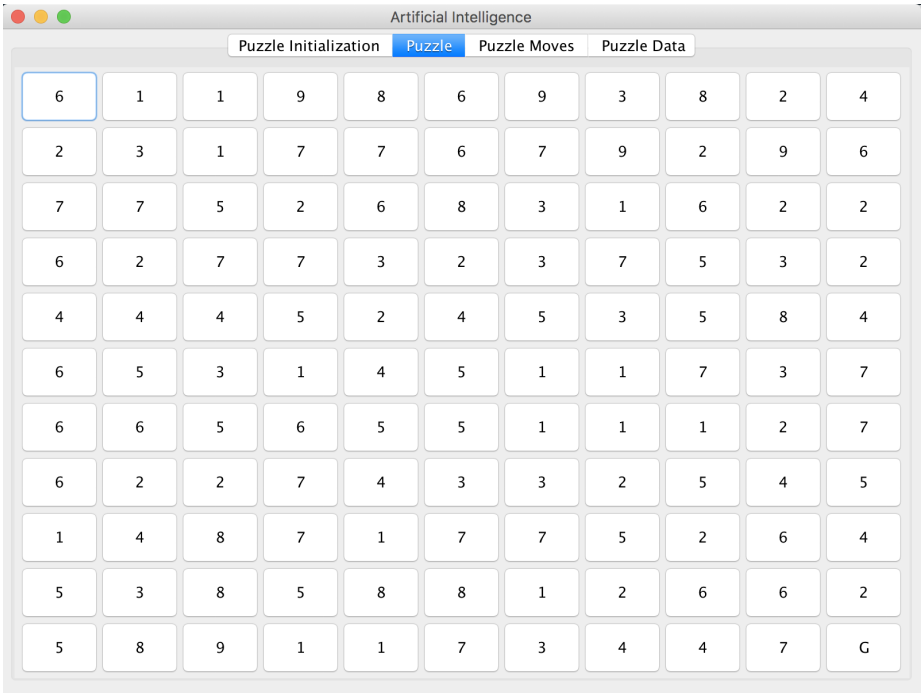

Figure 34: Unreachable Goal Puzzle size n = 11

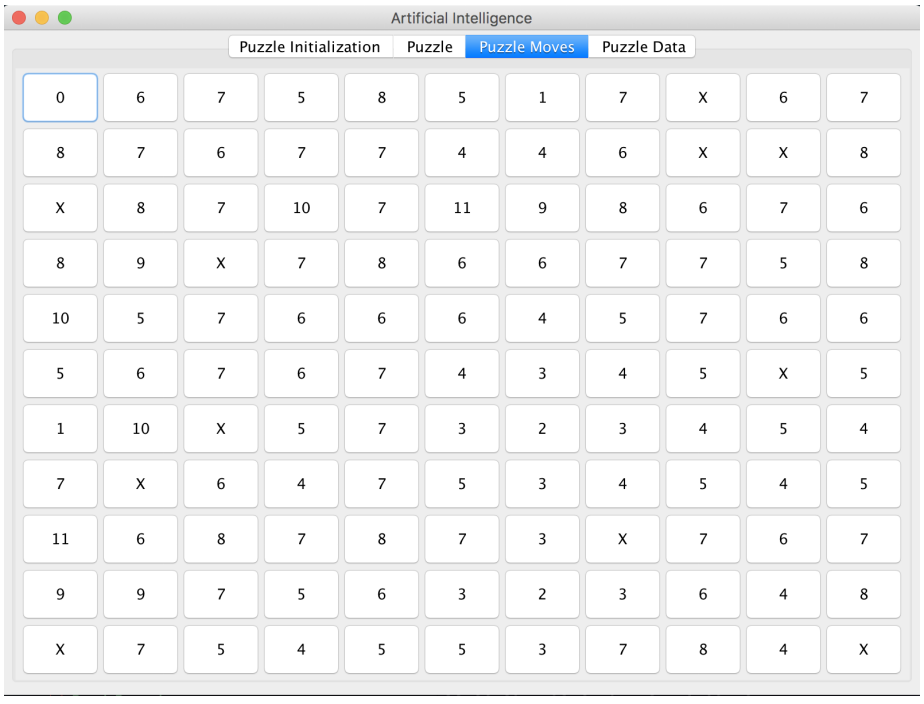

Figure 35: Unreachable Goal Puzzle Moves size n = 11

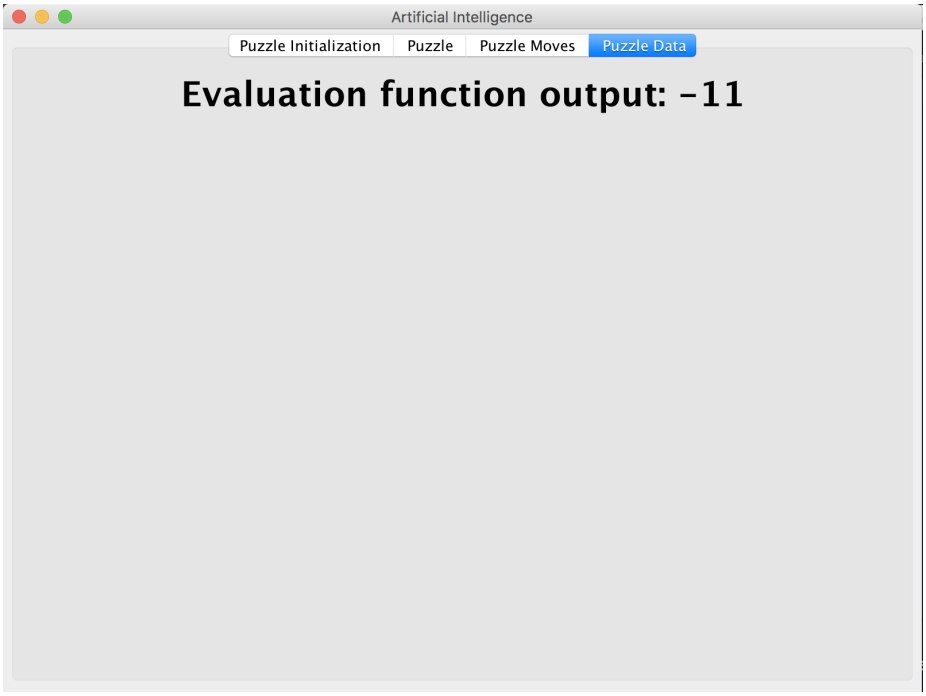

Figure 36: Unreachable Goal Puzzle Evaluation size n = 11

Basic Hill Climbing Approach

An Iteration of the Basic Hill Climbing Algorithm takes a random non goal cell and changes its value such that its legal and different from the previous value, the evaluation function is then run on the puzzle and the output compared to the previous evaluation output. If the result is the same or better than the new puzzle is accepted, otherwise the value is changed back to the previous value. The Basic Hill Climbing approach was run for a total of 100,000 iterations for each of the puzzle sizes. Sample puzzles along with graphs corresponding to 100,000 iterations averaged over 50 trials are included below.

Example Puzzle for n = 5

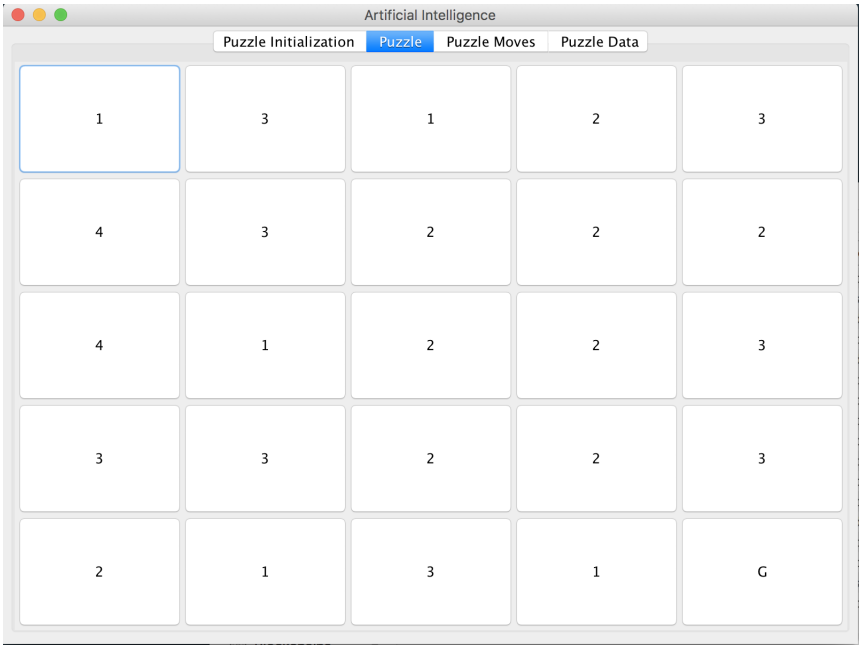

Figure 37: Basic Hill Climbing Best Puzzle after 100,000 iterations for n = 5

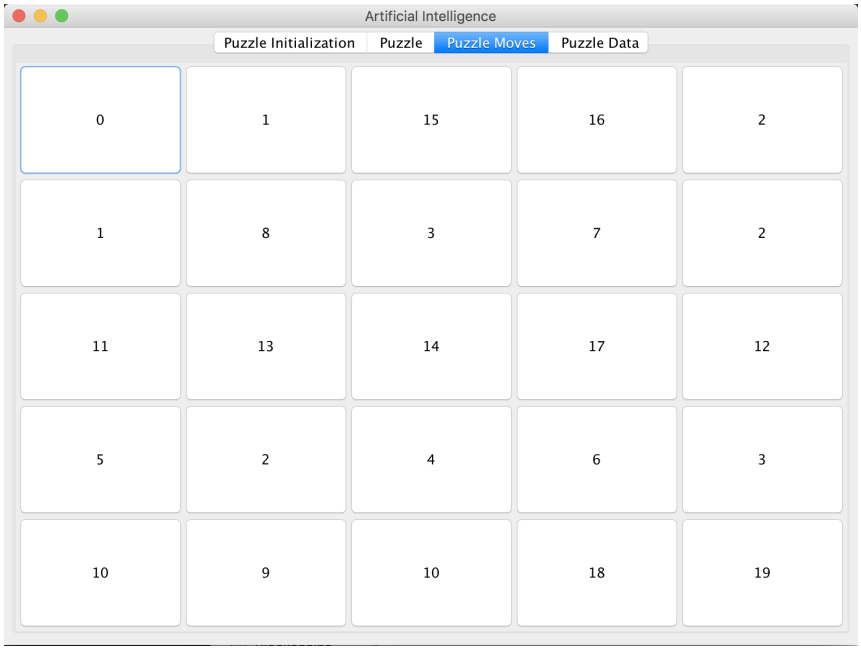

Figure 38: Basic Hill Climbing Puzzle Moves after 100,000 iterations for n = 5

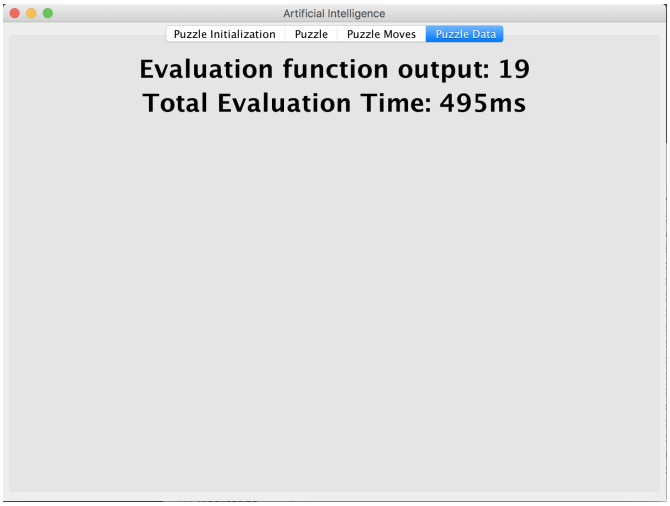

Figure 39: Basic Hill Climbing Puzzle Evaluation after 100,000 iterations for n = 5

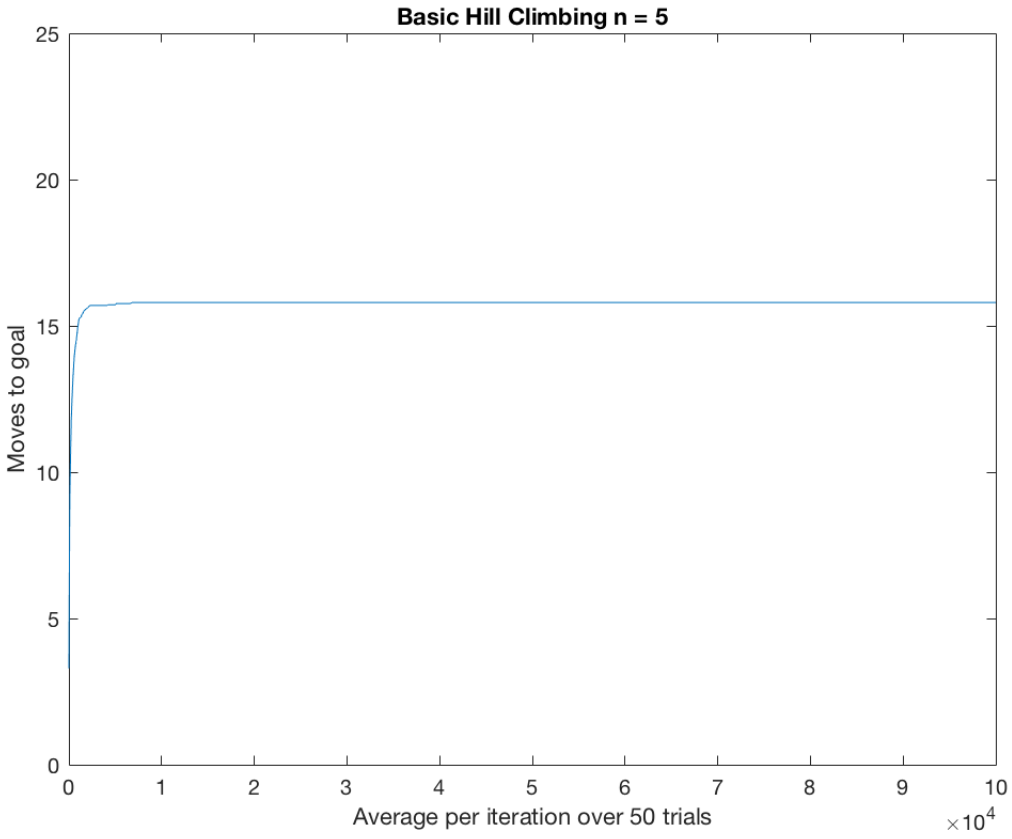

Figure 40: Plot of 100,000 iterations averaged over 50 runs for n = 5

Example Puzzle for n = 7

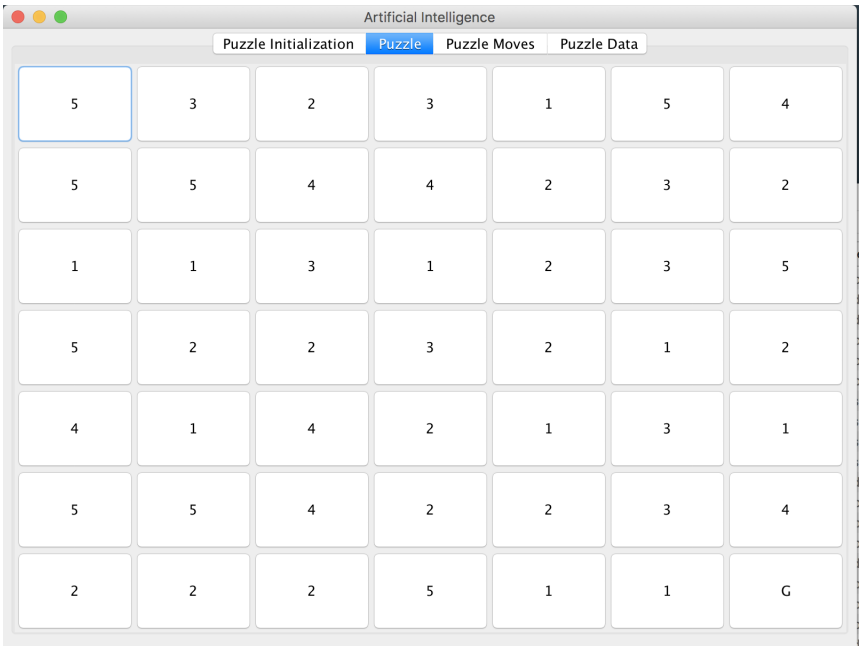

Figure 41: Basic Hill Climbing Best Puzzle after 100,000 iterations for n = 7

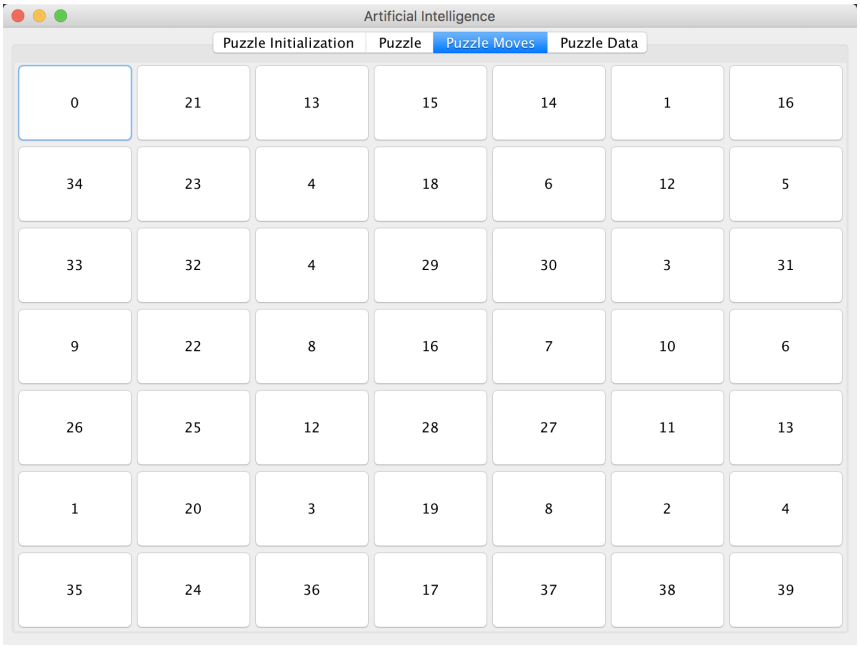

Figure 42: Basic Hill Climbing Puzzle Moves after 100,000 iterations for n = 7

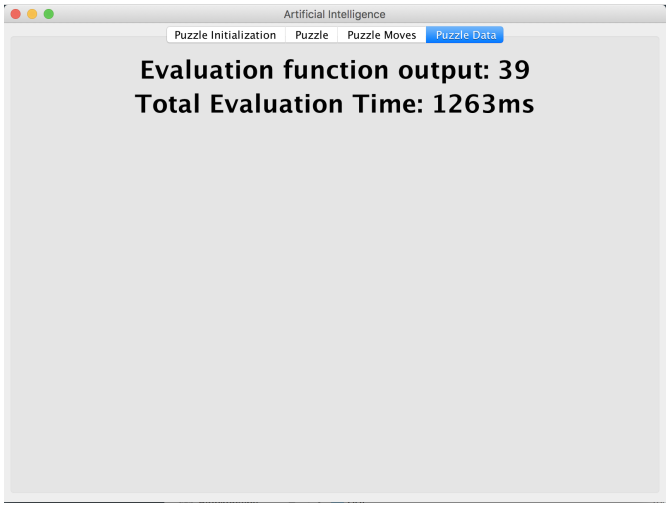

Figure 43: Basic Hill Climbing Puzzle Evaluation after 100,000 iterations for n = 7

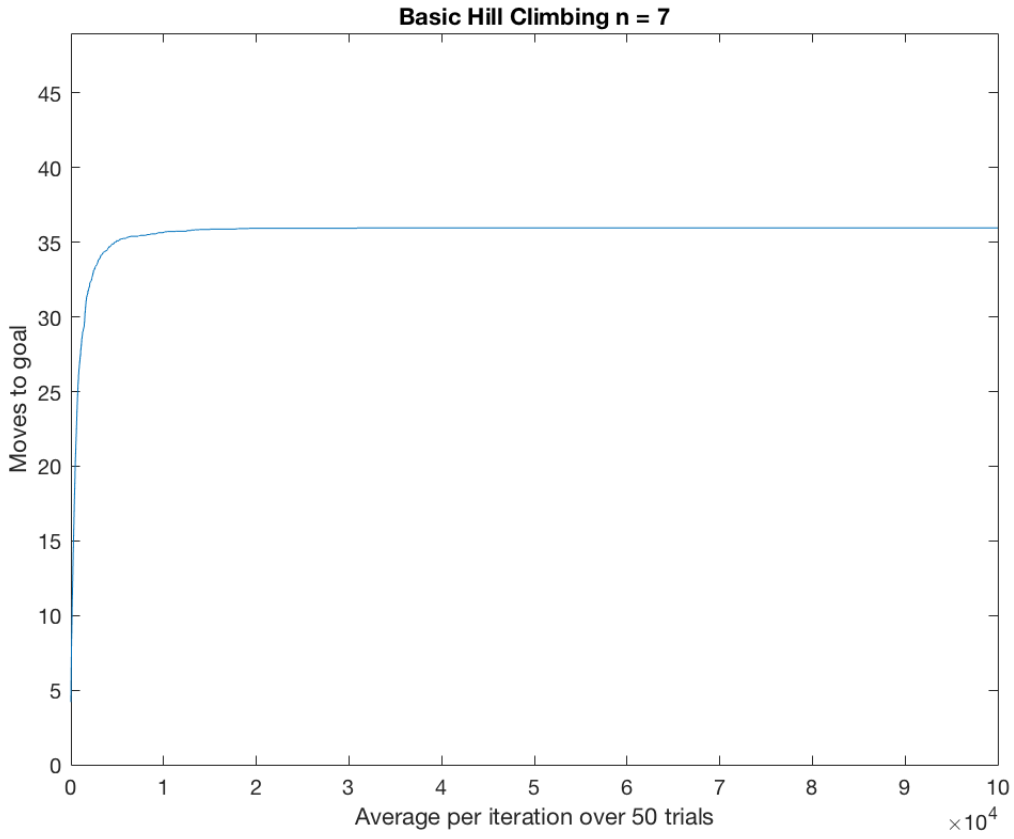

Figure 44: Plot of 100,000 iterations averaged over 50 runs for n = 7

Example Puzzle for n = 9

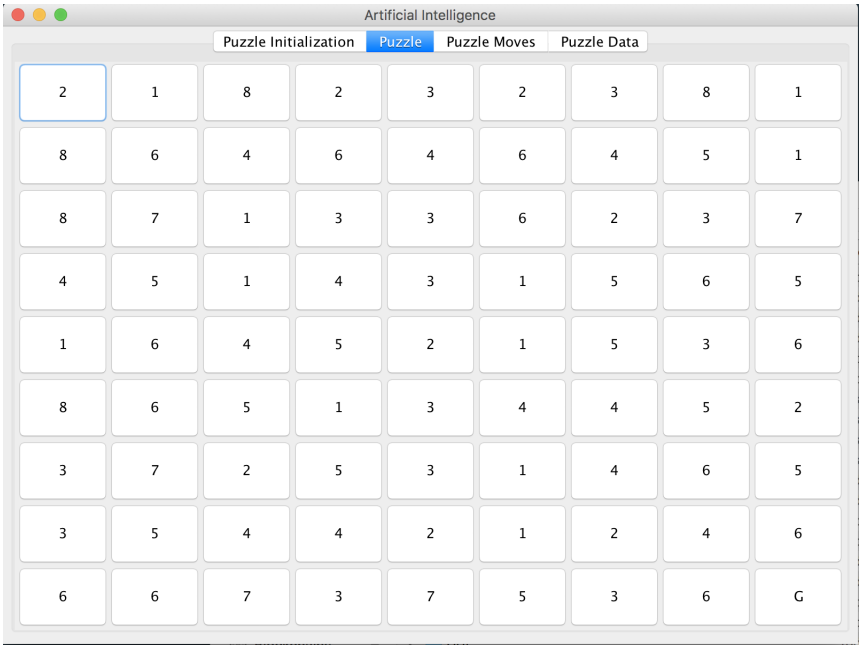

Figure 45: Basic Hill Climbing Best Puzzle after 100,000 iterations for n = 9

Figure 46: Basic Hill Climbing Puzzle Moves after 100,000 iterations for n = 9

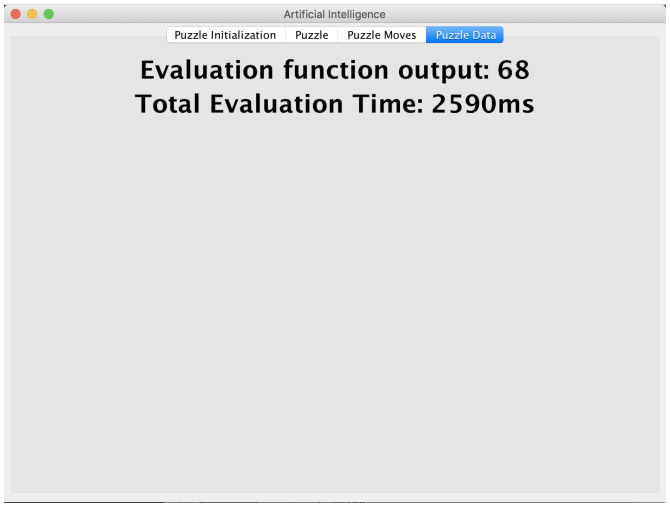

Figure 47: Basic Hill Climbing Puzzle Evaluation after 100,000 iterations for n = 9

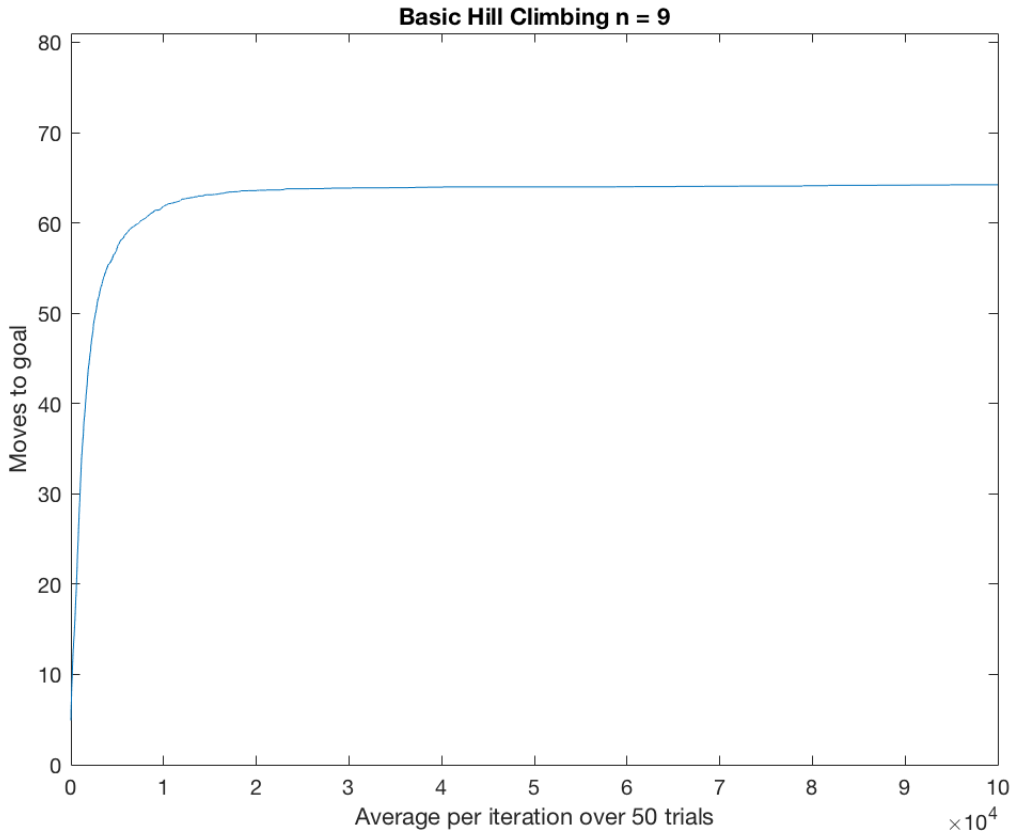

Figure 48: Plot of 100,000 iterations averaged over 50 runs for n = 9

Example Puzzle for n = 11

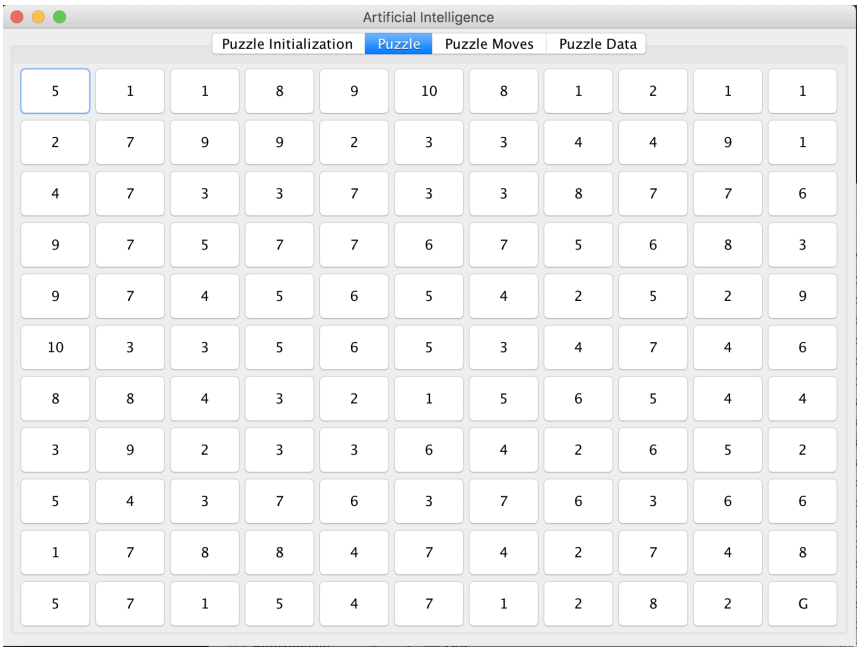

Figure 49: Basic Hill Climbing Best Puzzle after 100,000 iterations for n = 11

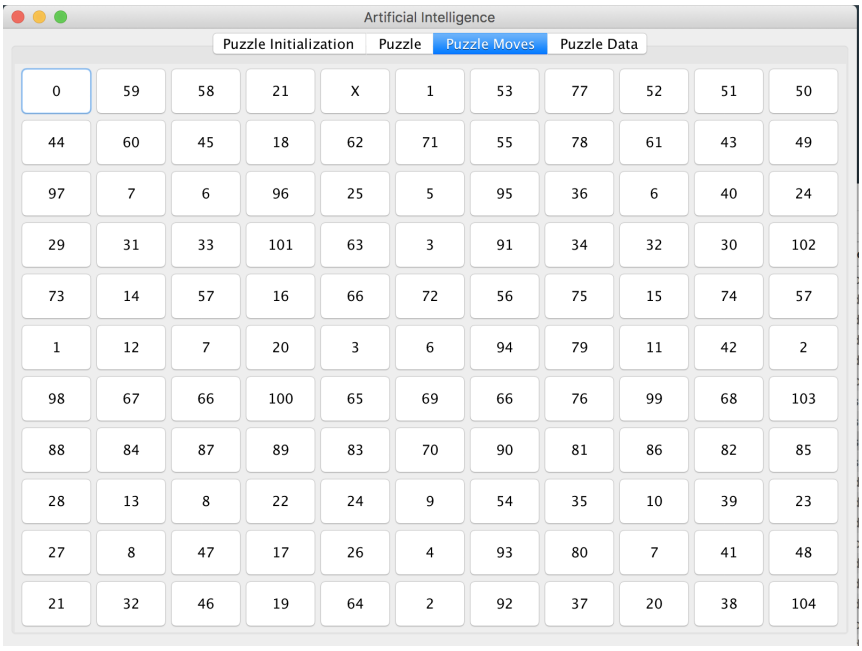

Figure 50: Basic Hill Climbing Puzzle Moves after 100,000 iterations for n = 11

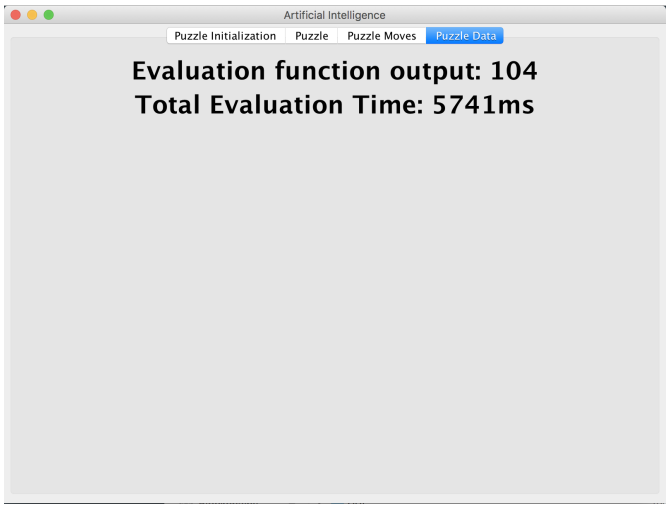

Figure 51: Basic Hill Climbing Puzzle Evaluation after 100,000 iterations for n = 11

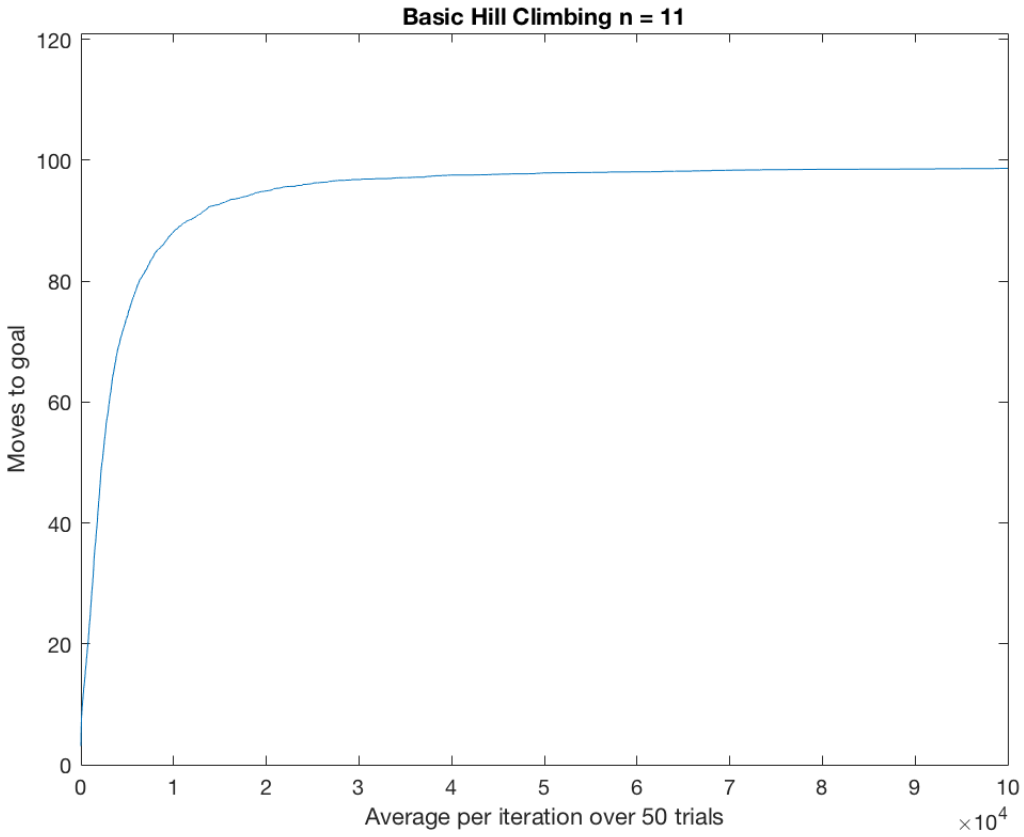

Figure 52: Plot of 100,000 iterations averaged over 50 runs for n = 11

Hill Climbing with Random Restarts

The Hill Climbing with Random Restarts approach aims to improve upon the Basic Hill Climbing approach by randomly restarting after n iterations. This is because the basic algorithm gets "stuck" in local maximum. The following puzzles were generated with 10,000 iterations and 10 restarts for a total of 100,000 iterations. Plots corresponding to the total 100,000 iterations averaged over 50 trials are included below. The results are similar to those of the standard hill climbing method.

Example Puzzle for n = 5

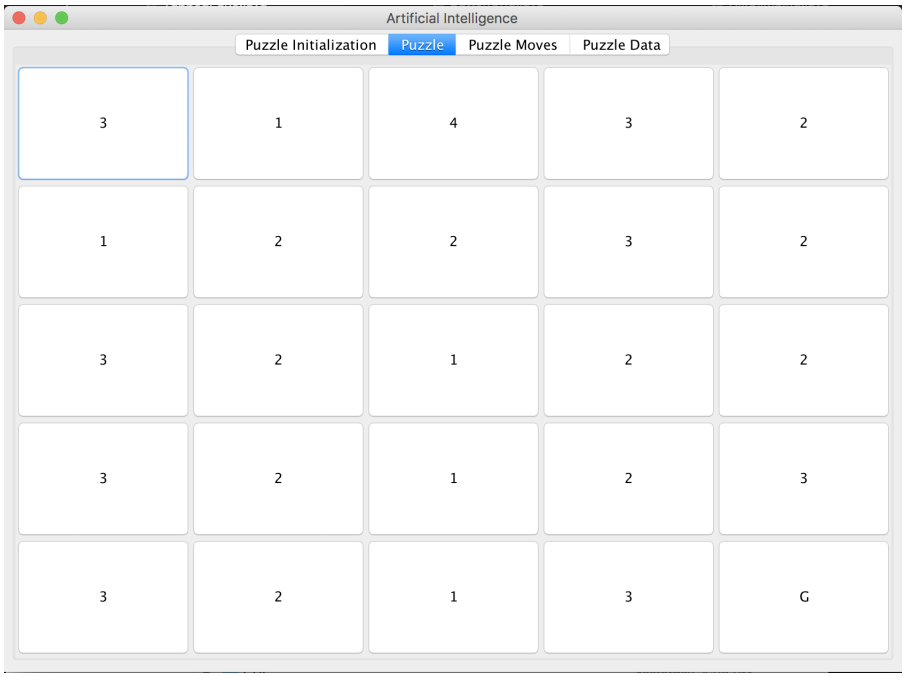

Figure 53: Hill Climbing with Random Restarts Best Puzzle after 100,000 iterations for n = 5

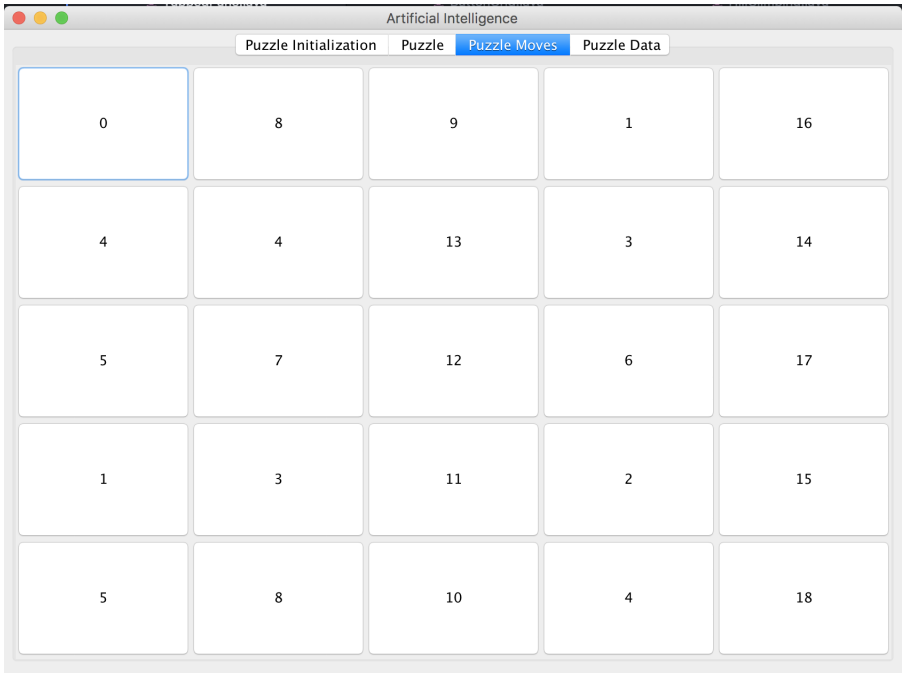

Figure 54: Hill Climbing with Random Restarts Best Puzzle Moves after 100,000 iterations for n = 5

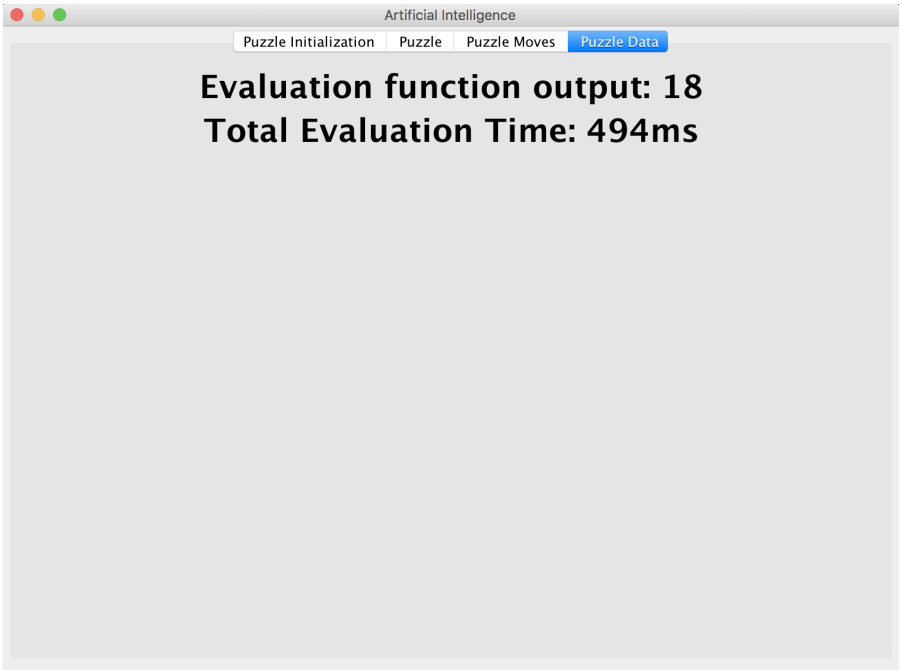

Figure 55: Hill Climbing with Random Restarts Best Puzzle Data after 100,000 iterations for n = 5

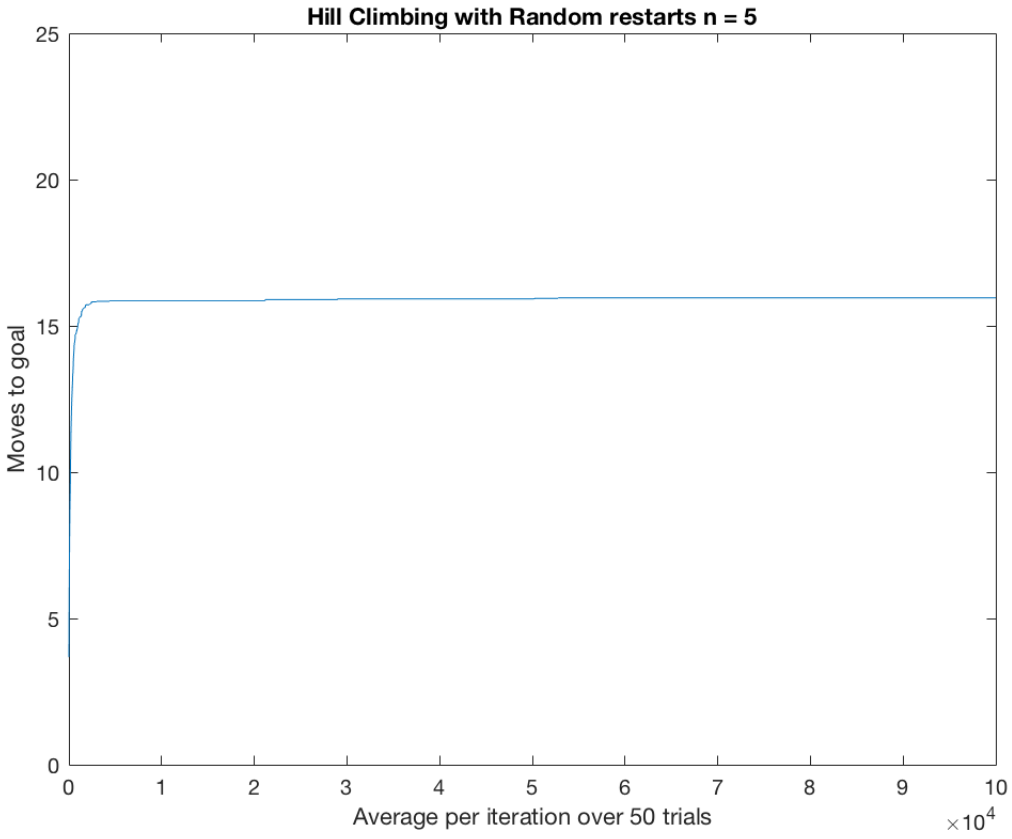

Figure 56: Plot of 100,000 iterations averaged over 50 runs for n = 5

Example Puzzle for n = 7

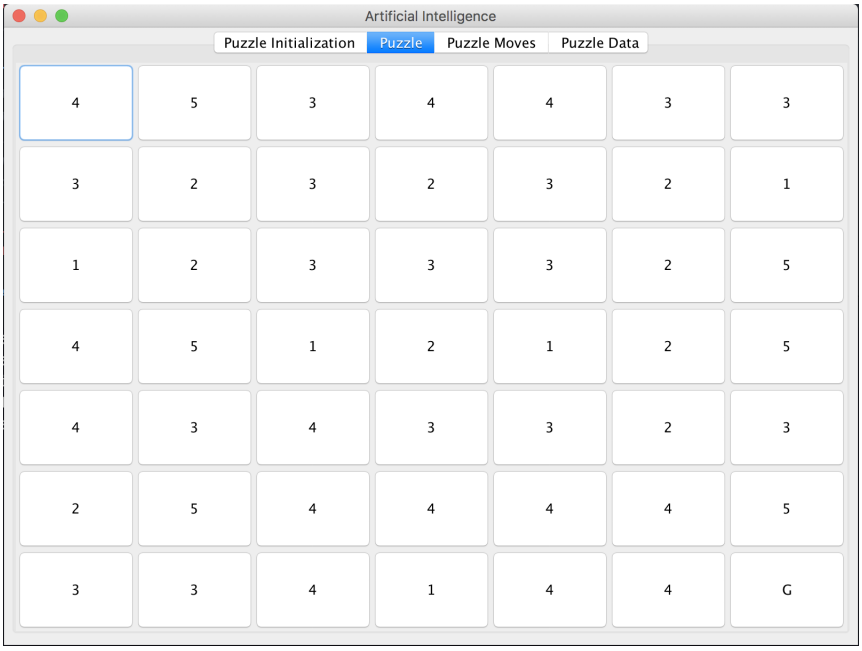

Figure 57: Hill Climbing with Random Restarts Best Puzzle after 100,000 iterations for n = 7

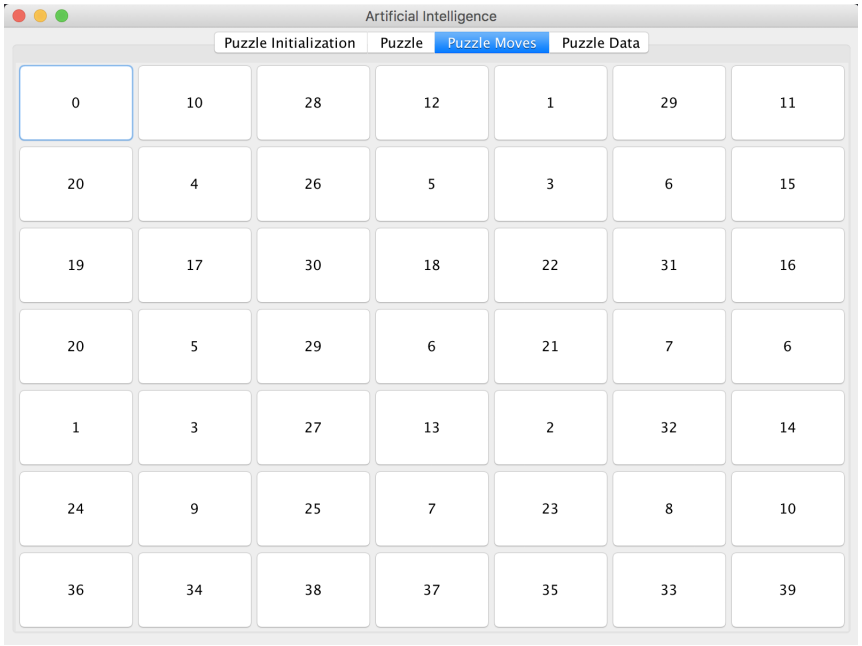

Figure 58: Hill Climbing with Random Restarts Best Puzzle Moves after 100,000 iterations for n = 7

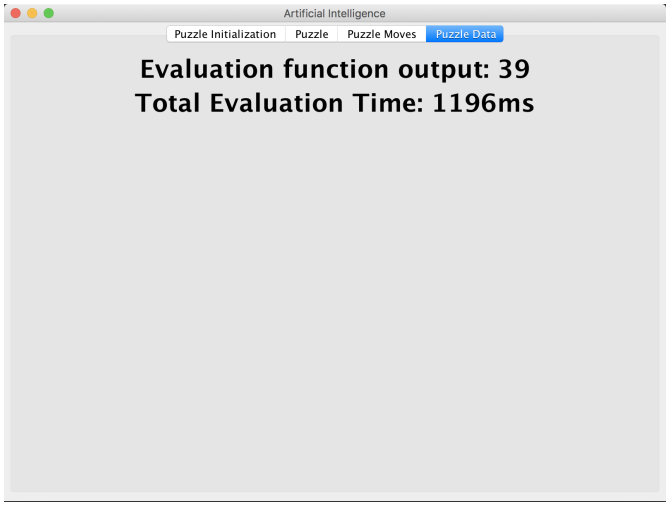

Figure 59: Hill Climbing with Random Restarts Best Puzzle Evaluation after 100,000 iterations for n = 7

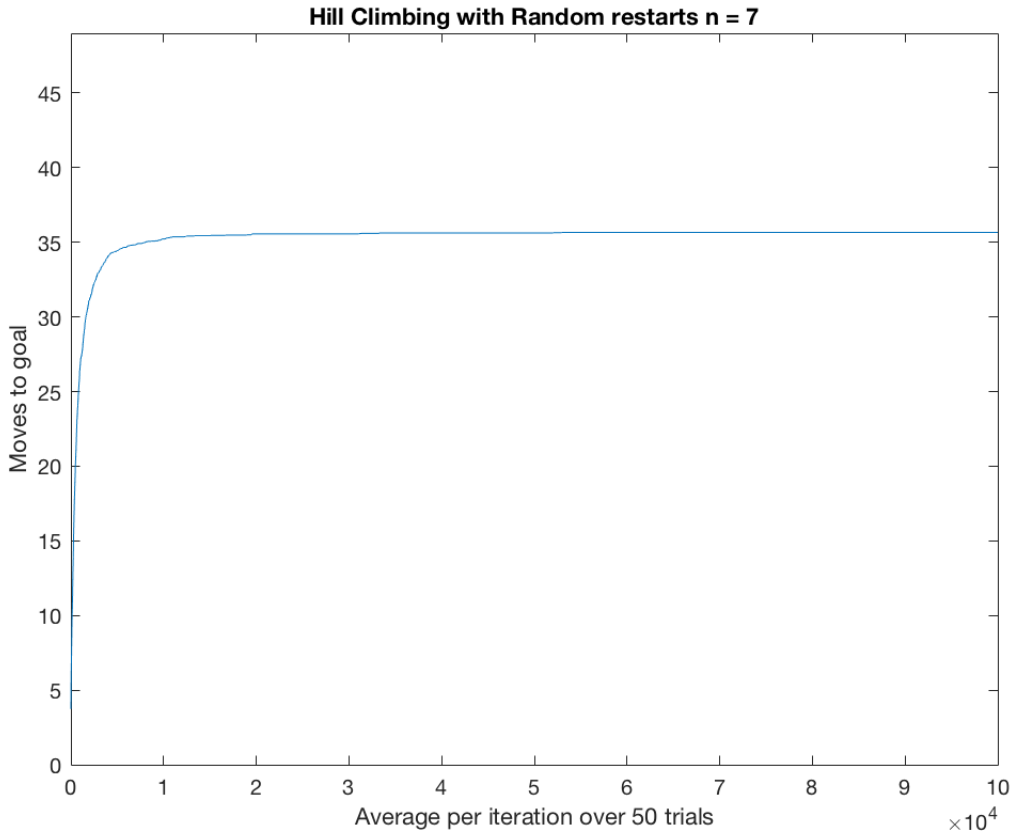

Figure 60: Plot of 100,000 iterations averaged over 50 runs for n = 7

Example Puzzle for n = 9

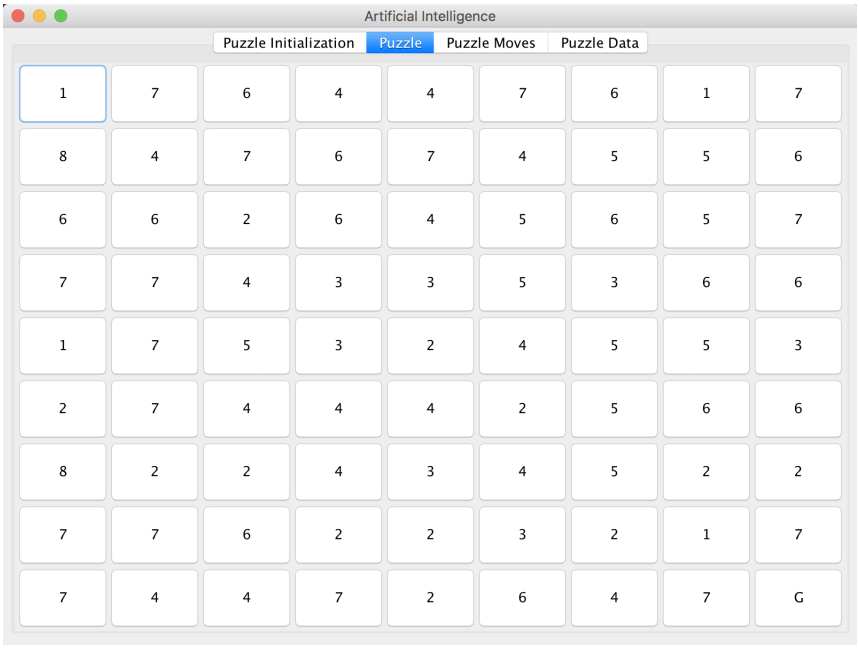

Figure 61: Hill Climbing with Random Restarts Best Puzzle after 100,000 iterations for n = 9

Figure 62: Hill Climbing with Random Restarts Best Puzzle Moves after 100,000 iterations for n = 9

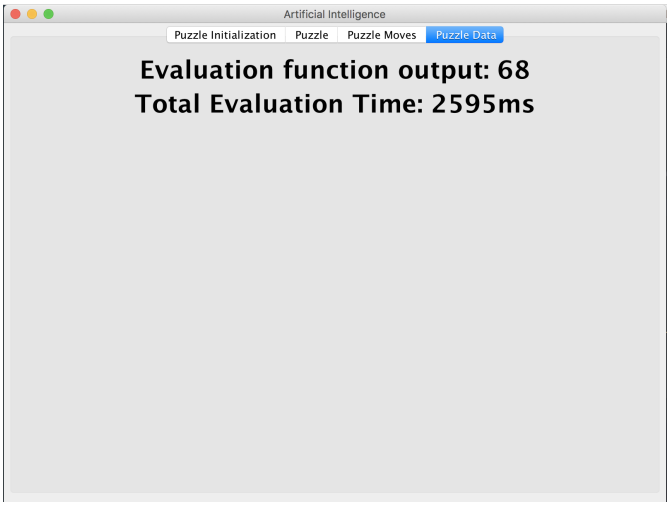

Figure 63: Hill Climbing with Random Restarts Best Puzzle Data after 100,000 iterations for n = 9

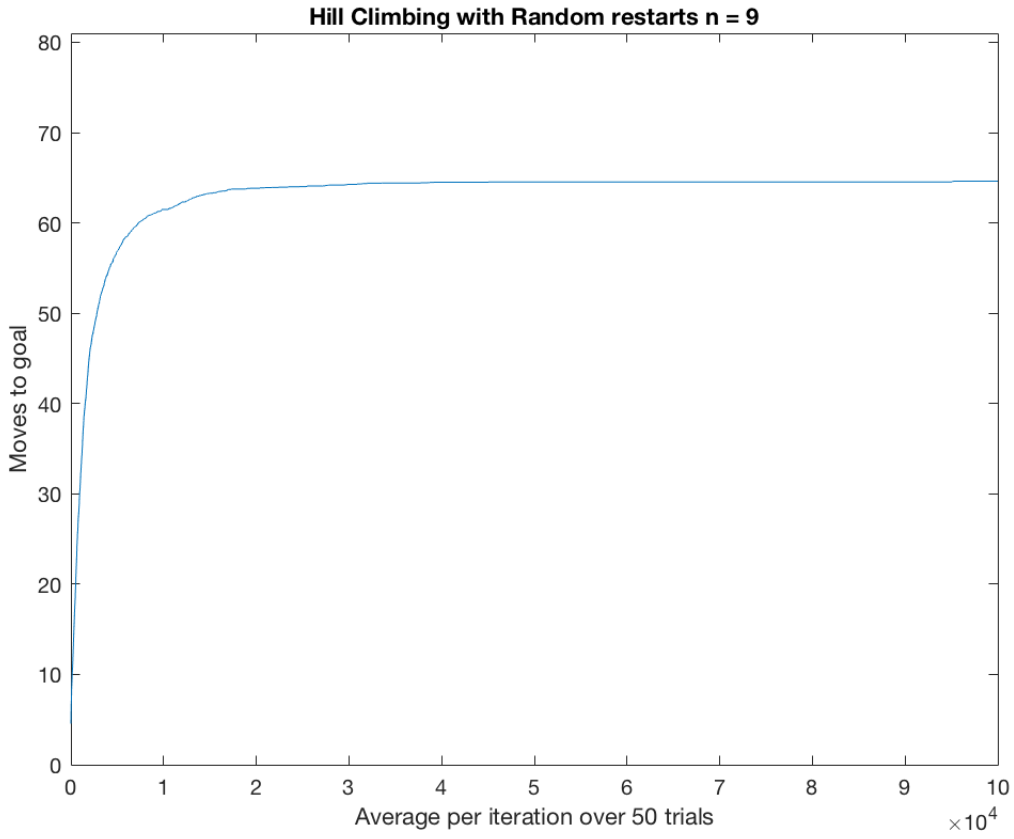

Figure 64: Plot of 100,000 iterations averaged over 50 runs for n = 9

Example Puzzle for n = 11

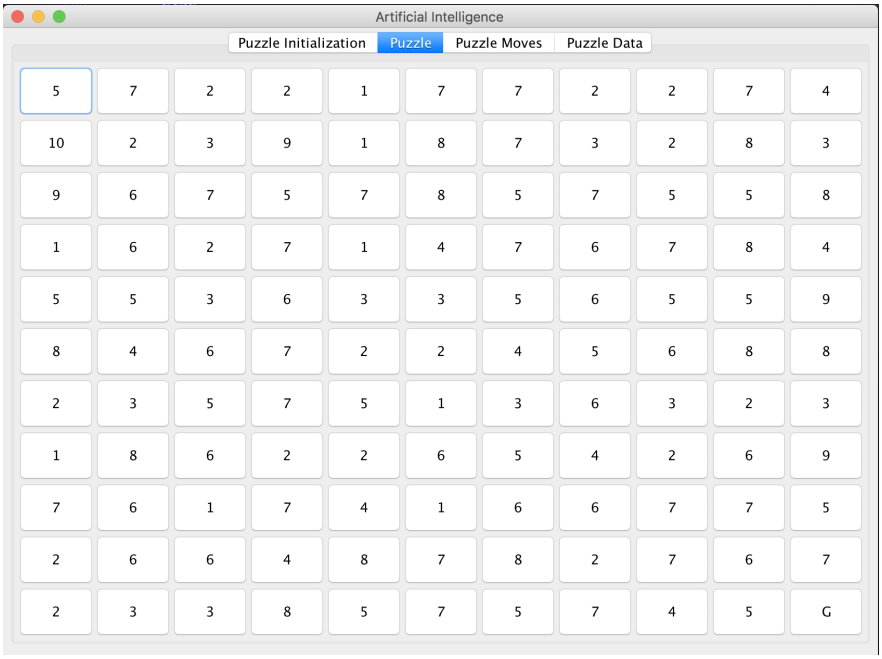

Figure 65: Hill Climbing with Random Restarts Best Puzzle after 100,000 iterations for n = 11

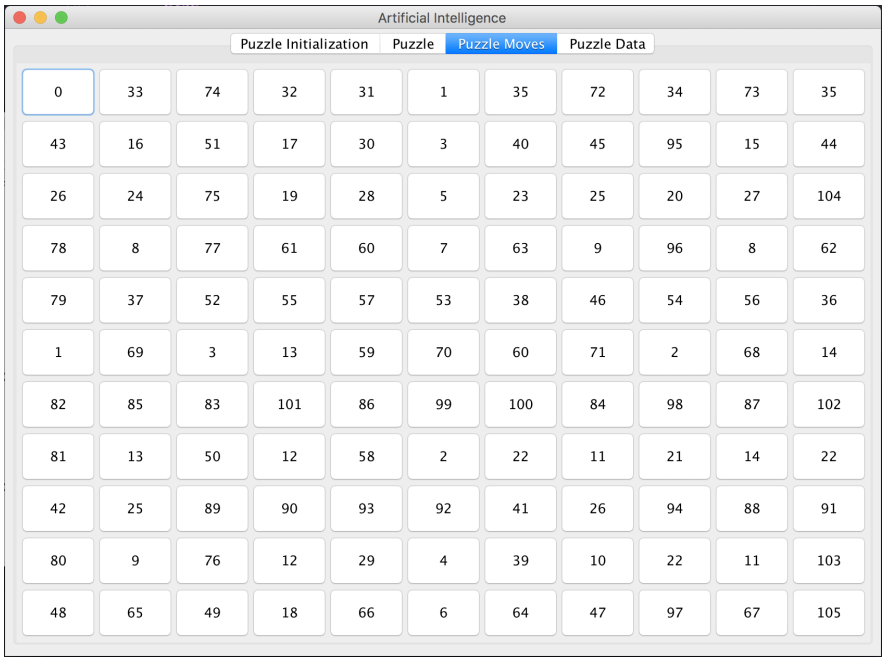

Figure 66: Hill Climbing with Random Restarts Best Puzzle Moves after 100,000 iterations for n = 11

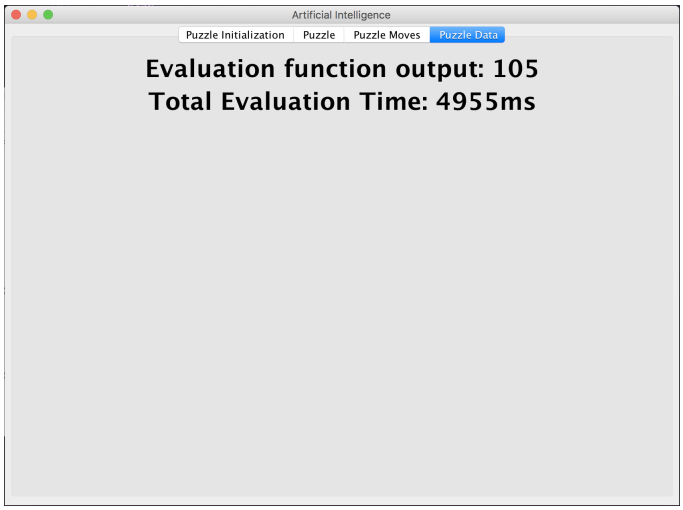

Figure 67: Hill Climbing with Random Restarts Best Puzzle Data after 100,000 iterations for n = 11

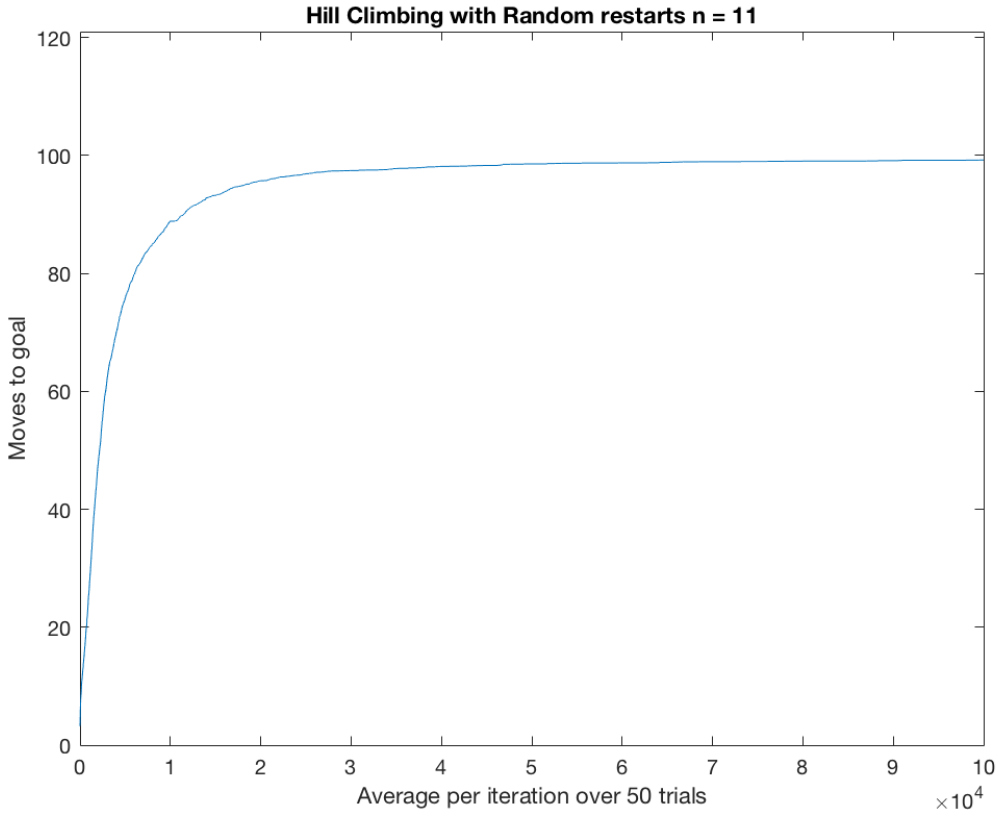

Figure 68: Plot of 100,000 iterations averaged over 50 runs for n = 11

Hill Climbing with Random Walks

There were a total of 100,000 iterations run for each of the puzzle sizes with a probability of lower acceptance set at 0.03. After running the algorithm for different values of p, probability of lower acceptance, it was found that as the probability of lower acceptance rose the output of the evaluation function lowered. The optimum choice for probability seemed to be within the 0.01-0.05 range.

Example Puzzle for n = 5

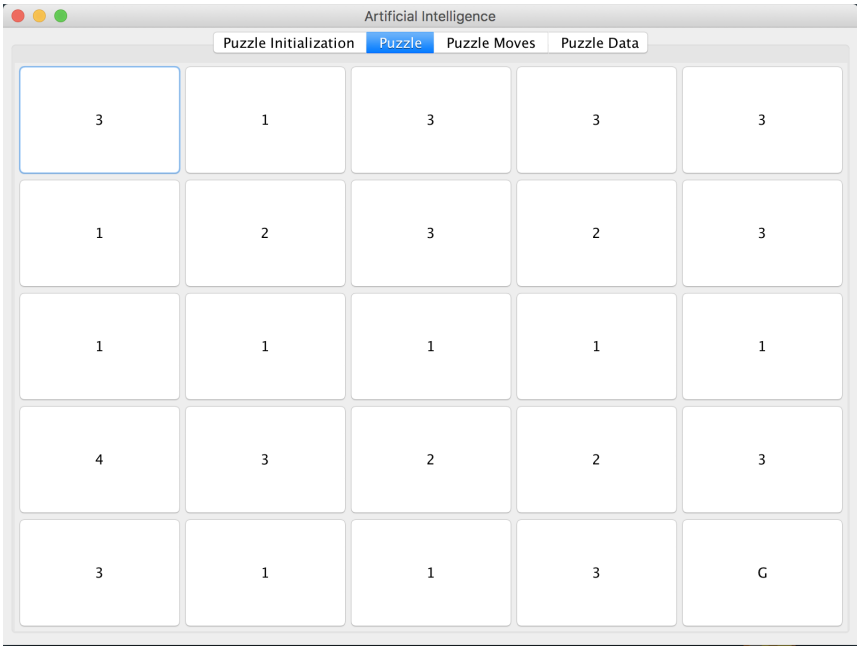

Figure 69: Hill Climbing with Random Walk Best Puzzle after 100,000 iterations for n = 5

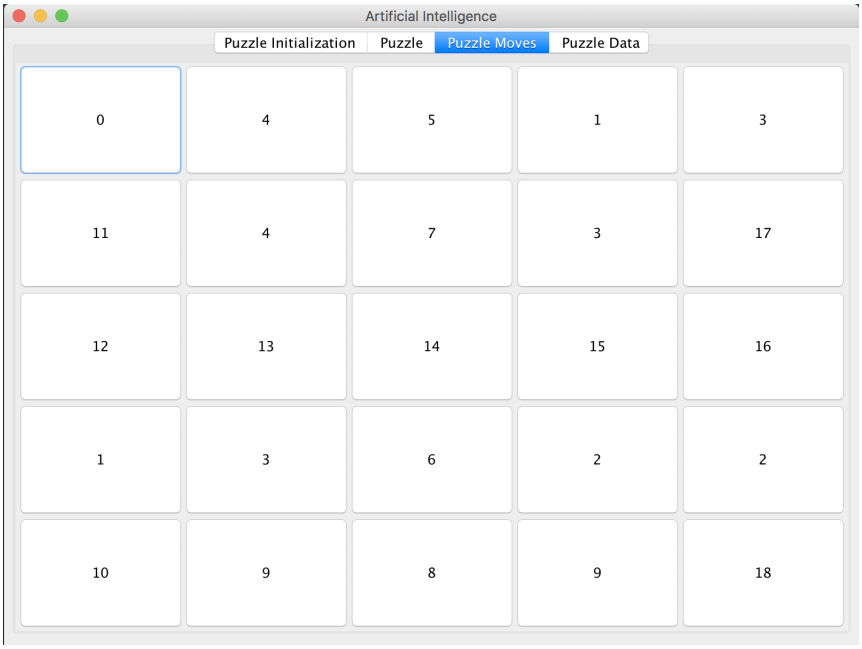

Figure 70: Hill Climbing with Random Walk Best Puzzle Moves after 100,000 iterations for n = 5

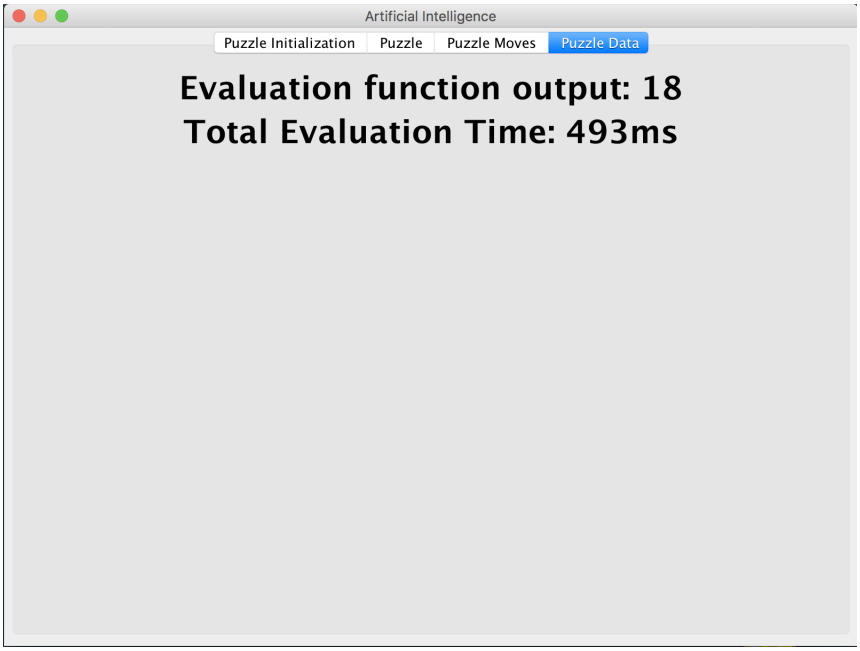

Figure 71: Hill Climbing with Random Walk Best Puzzle Data after 100,000 iterations for n = 5

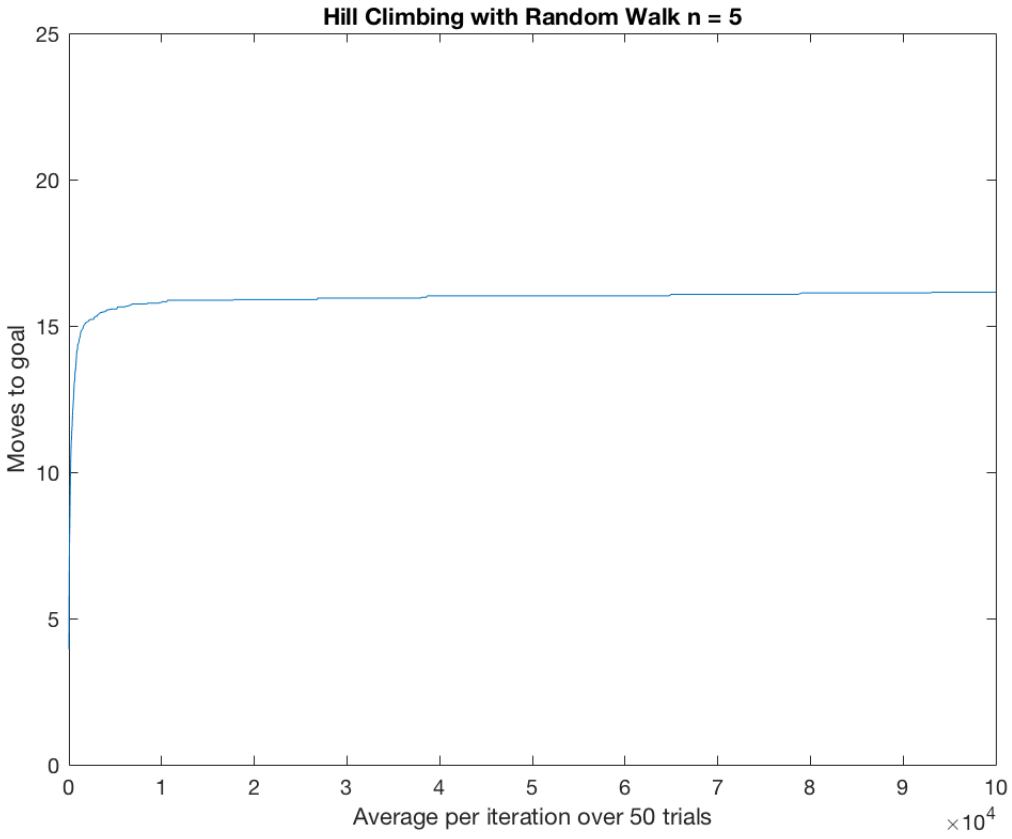

Figure 72: Plot of 100,000 iterations averaged over 50 runs for n = 5

Example Puzzle for n = 7

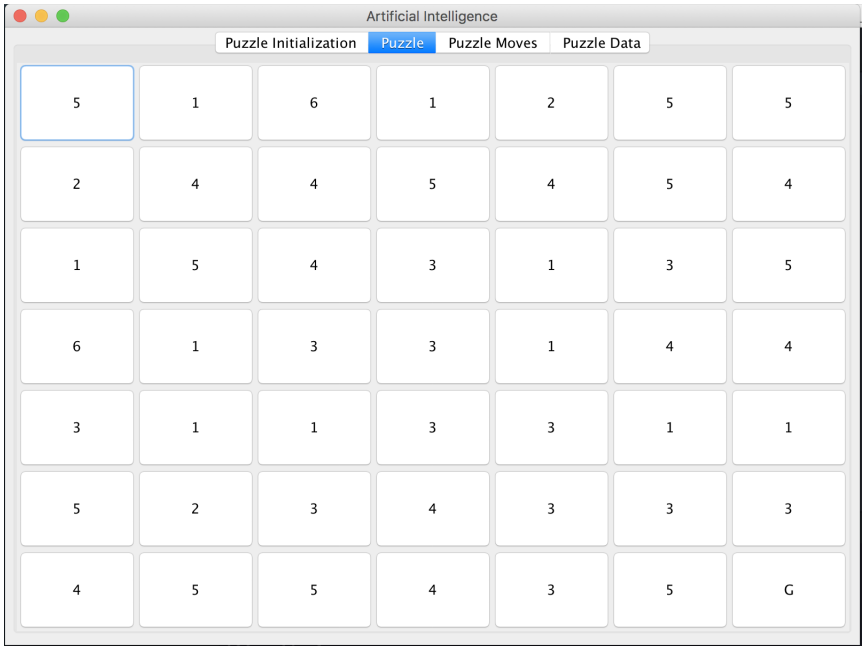

Figure 73: Hill Climbing with Random Walk Best Puzzle after 100,000 iterations for n = 7

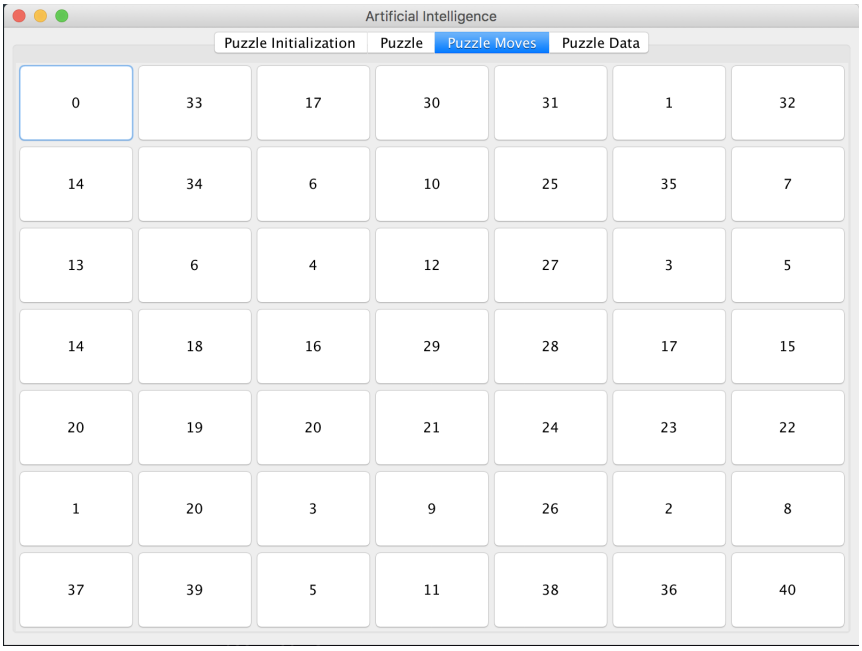

Figure 74: Hill Climbing with Random Walk Best Puzzle Moves after 100,000 iterations for n = 7

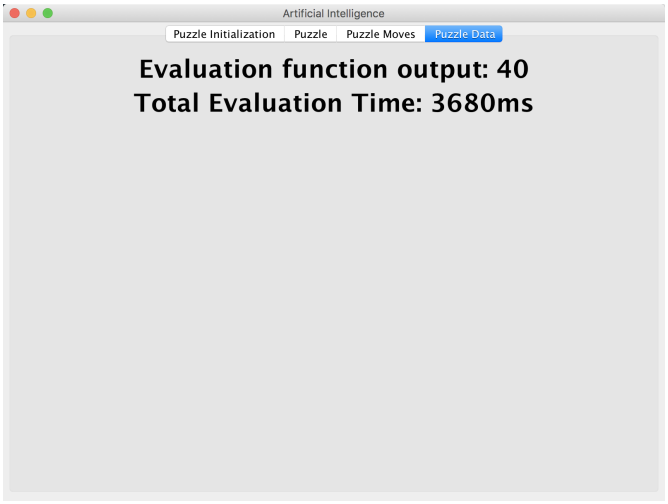

Figure 75: Hill Climbing with Random Walk Best Puzzle Data after 100,000 iterations for n = 7

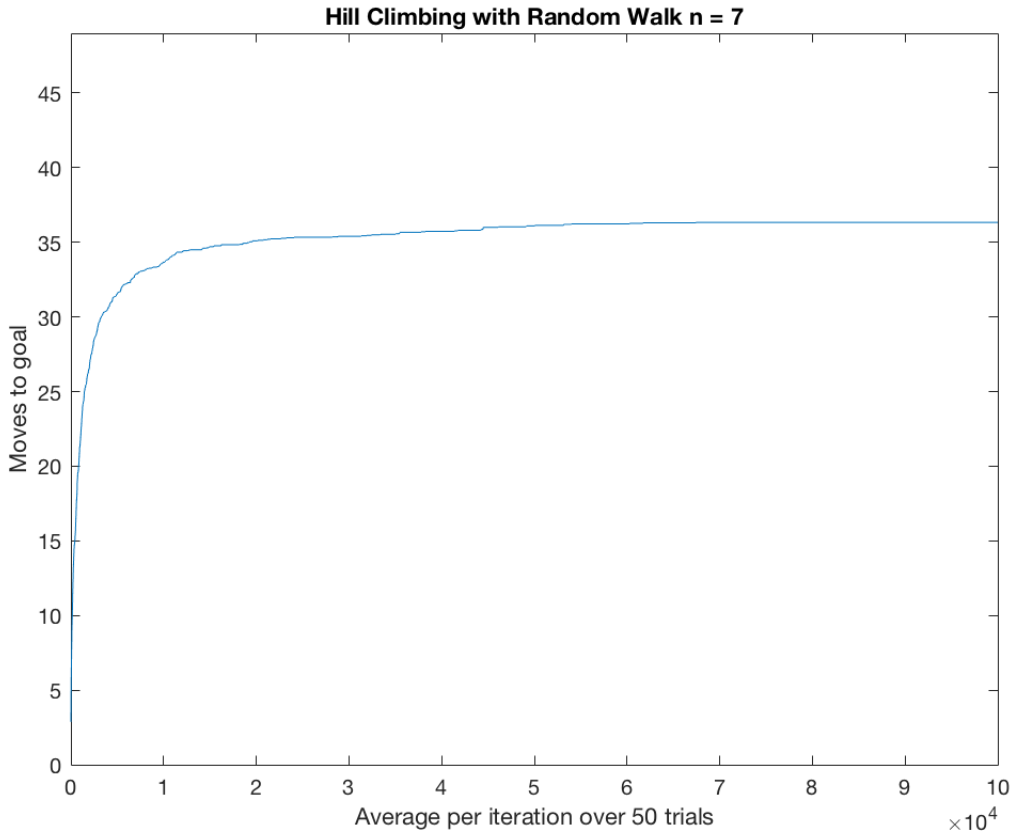

Figure 76: Plot of 100,000 iterations averaged over 50 runs for n = 7

Example Puzzle for n = 9

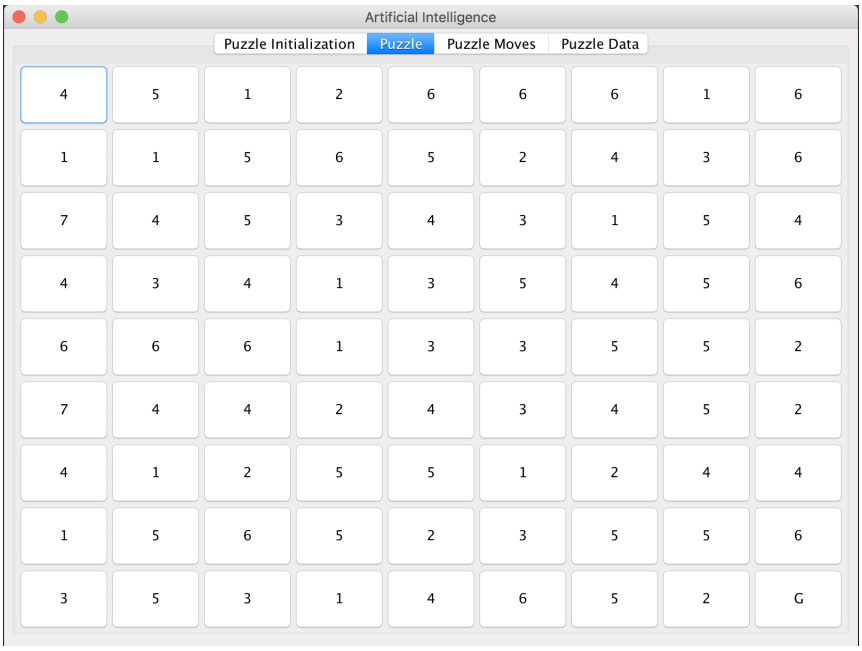

Figure 77: Hill Climbing with Random Walk Best Puzzle after 100,000 iterations for n = 9

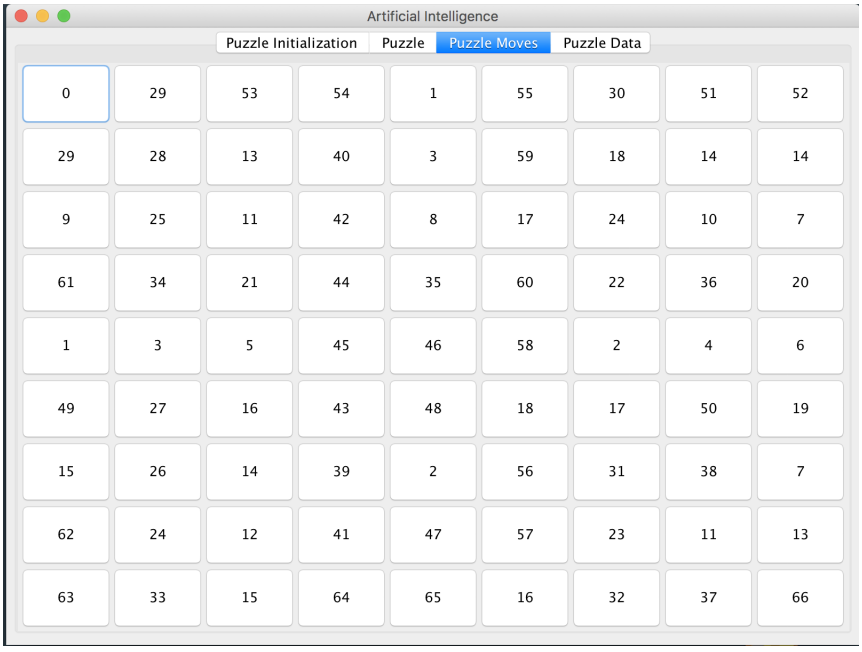

Figure 78: Hill Climbing with Random Walk Best Puzzle Moves after 100,000 iterations for n = 9

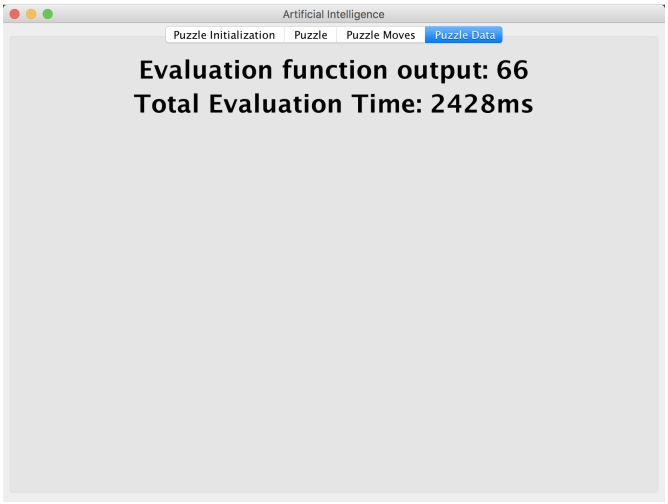

Figure 79: Hill Climbing with Random Walk Best Puzzle Data after 100,000 iterations for n = 9

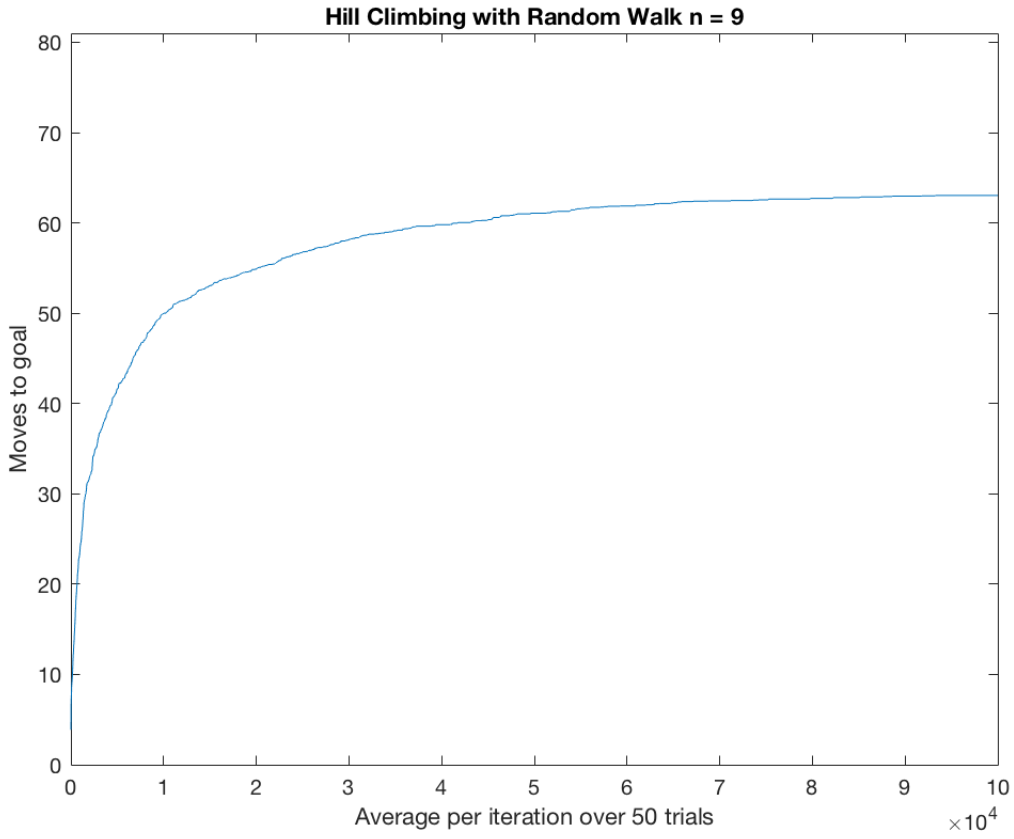

Figure 80: Plot of 100,000 iterations averaged over 50 runs for n = 9

Example Puzzle for n = 11

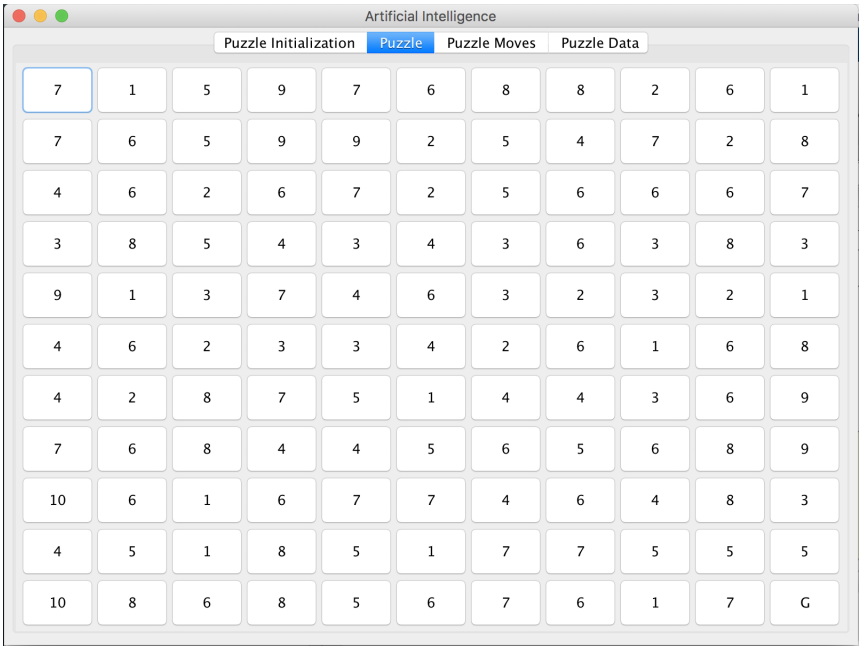

Figure 81: Hill Climbing with Random Walk Best Puzzle after 100,000 iterations for n = 11

Figure 82: Hill Climbing with Random Walk Best Puzzle Moves after 100,000 iterations for n = 11

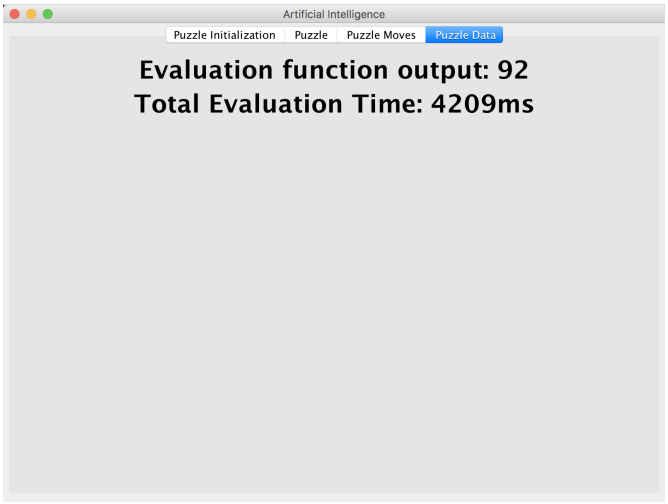

Figure 83: Hill Climbing with Random Walk Best Puzzle Data after 100,000 iterations for n = 11

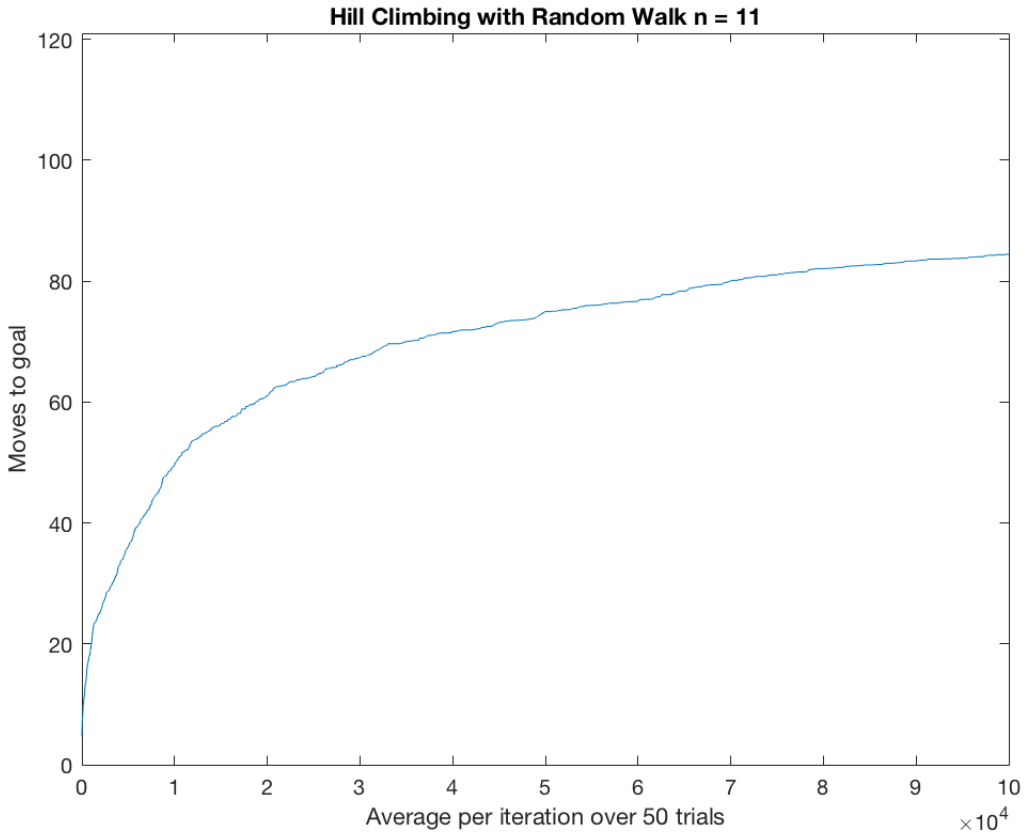

Figure 84: Plot of 100,000 iterations averaged over 50 runs for n = 11

Simulated Annealing

Example Puzzle for n = 5

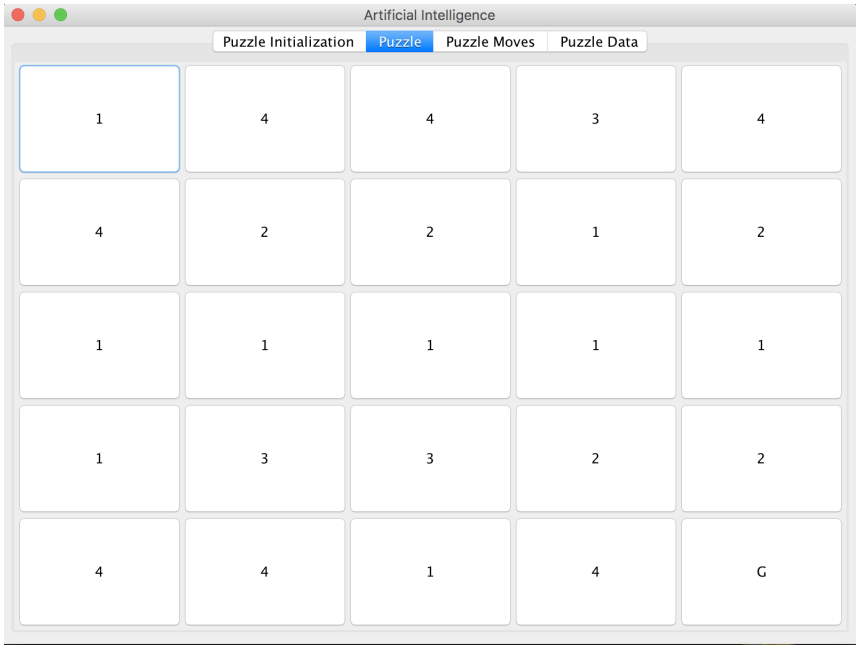

Figure 85: Simulated Annealing Best Puzzle after 100,000 iterations for n = 5

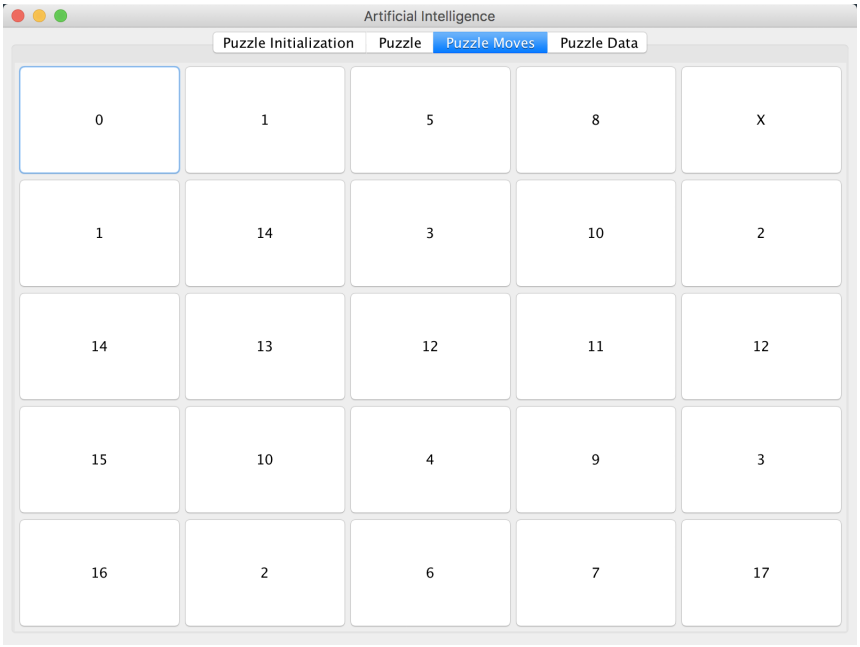

Figure 86: Simulated Annealing Best Puzzle Moves after 100,000 iterations for n = 5

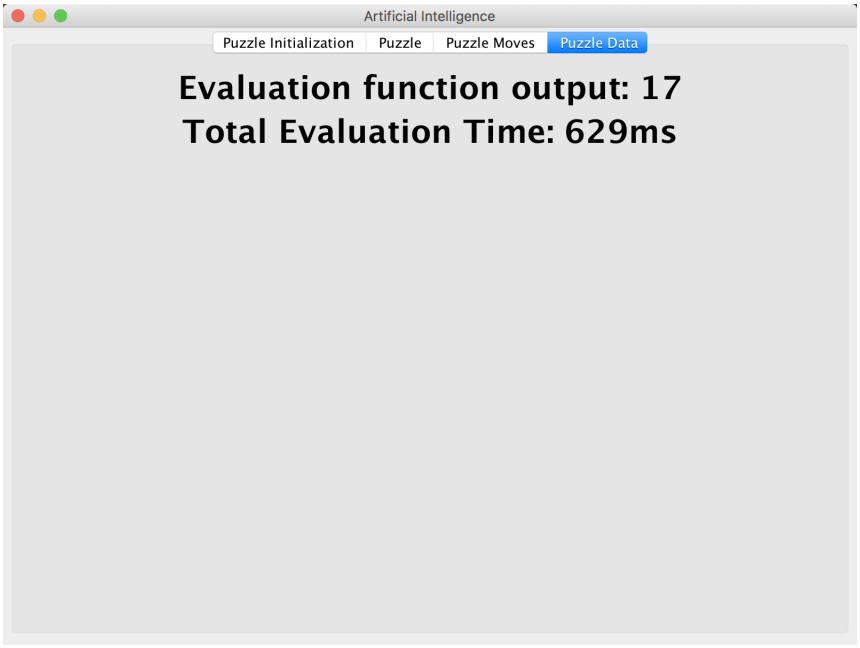

Figure 87: Simulated Annealing Best Puzzle Evaluation after 100,000 iterations for n = 5

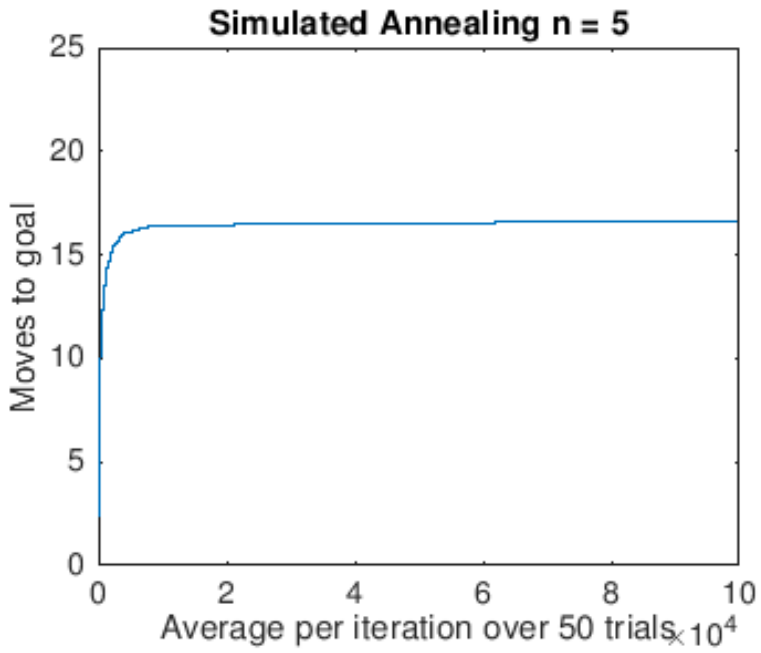

Figure 88: Plot of 100,000 iterations averaged over 50 runs for n = 5

Example Puzzle for n = 7

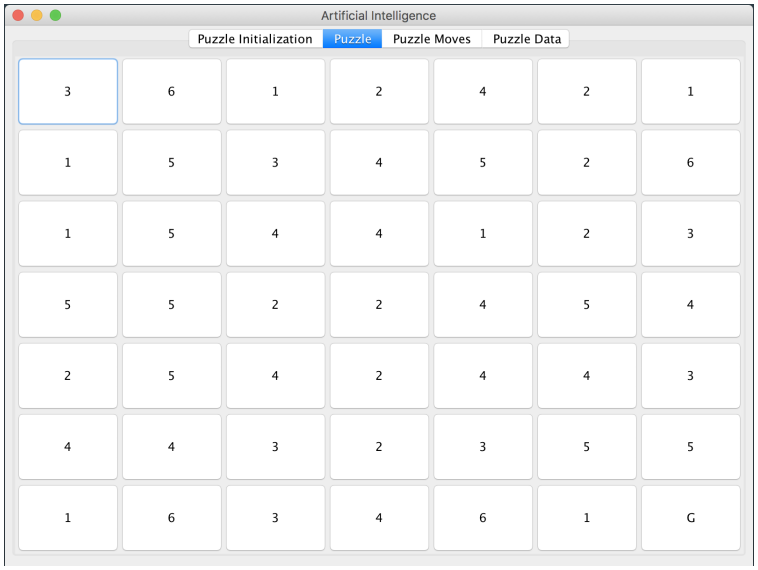

Figure 89: Simulated Annealing Best Puzzle after 100,000 iterations for n = 7

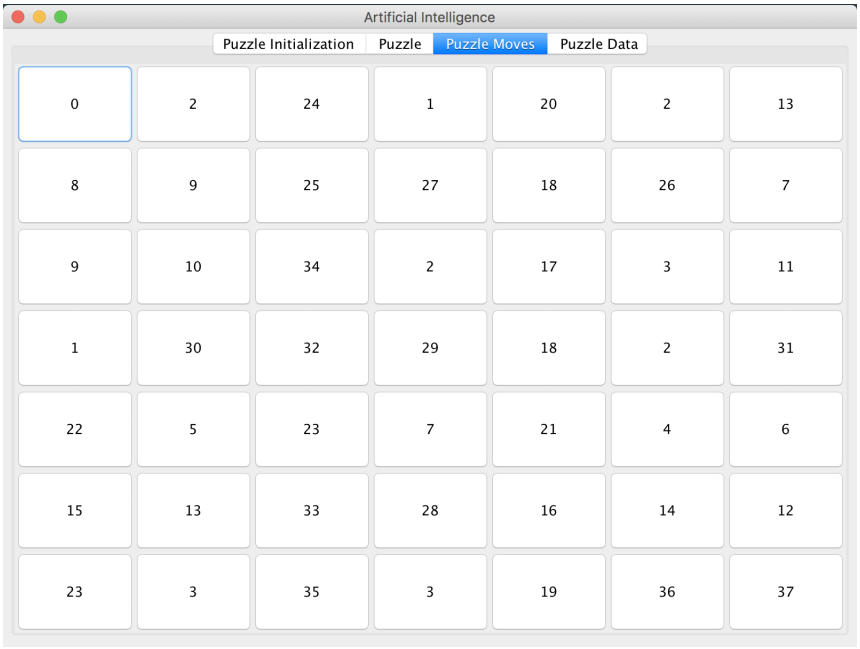

Figure 90: Simulated Annealing Best Puzzle Moves after 100,000 iterations for n = 7

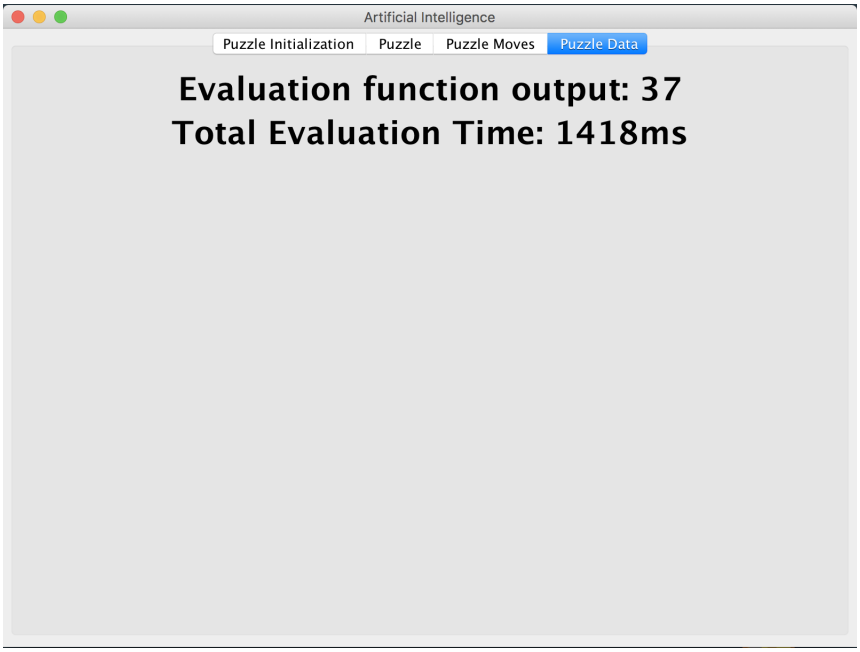

Figure 91: Simulated Annealing Best Puzzle Evaluation after 100,000 iterations for n = 7

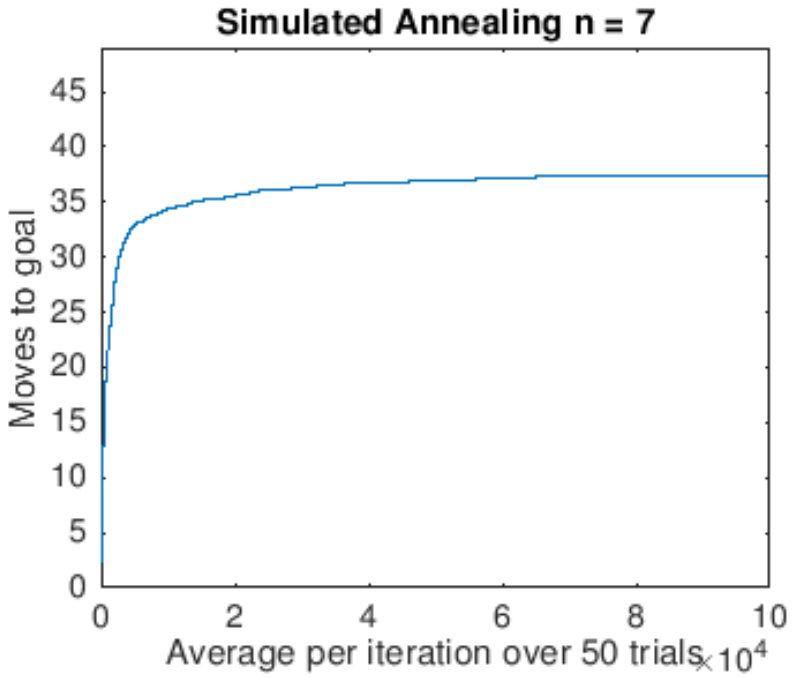

Figure 92: Plot of 100,000 iterations averaged over 50 runs for n = 7

Example Puzzle for n = 9

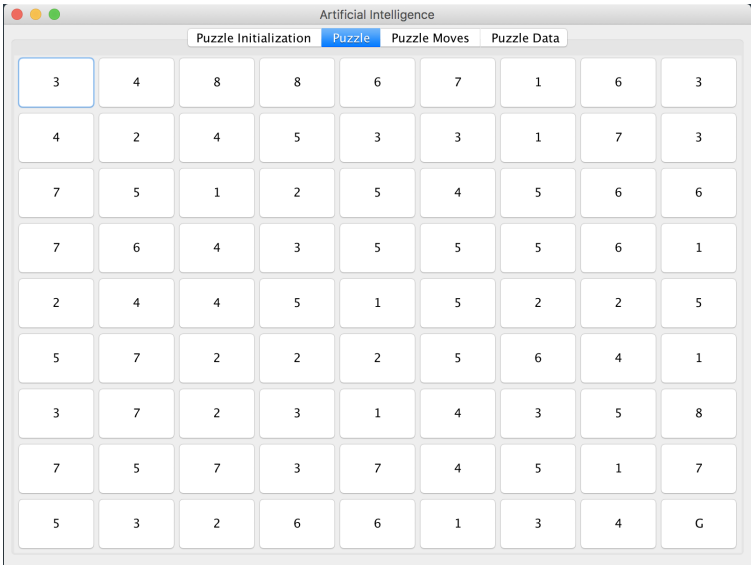

Figure 93: Simulated Annealing Best Puzzle after 100,000 iterations for n = 9

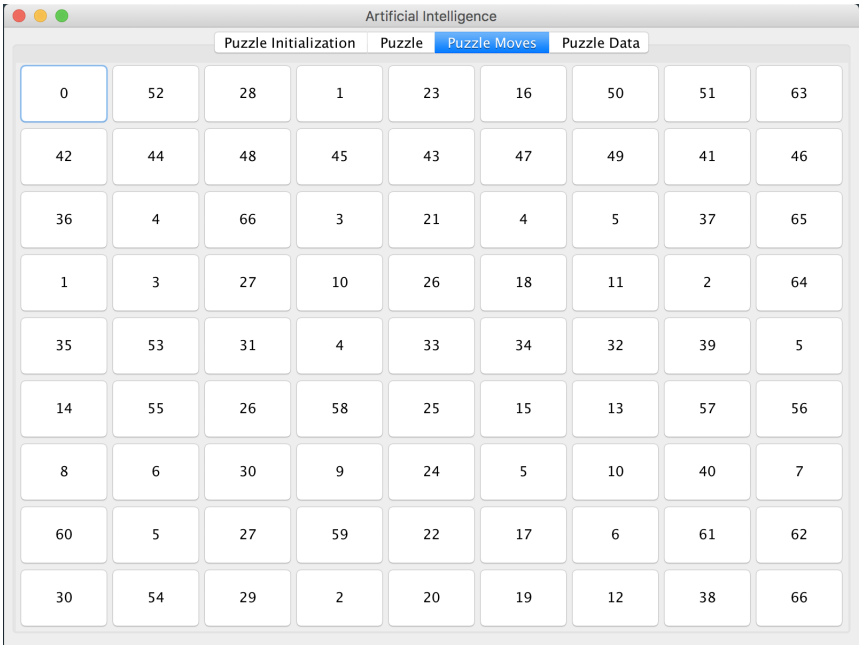

Figure 94: Simulated Annealing Best Puzzle Moves after 100,000 iterations for n = 9

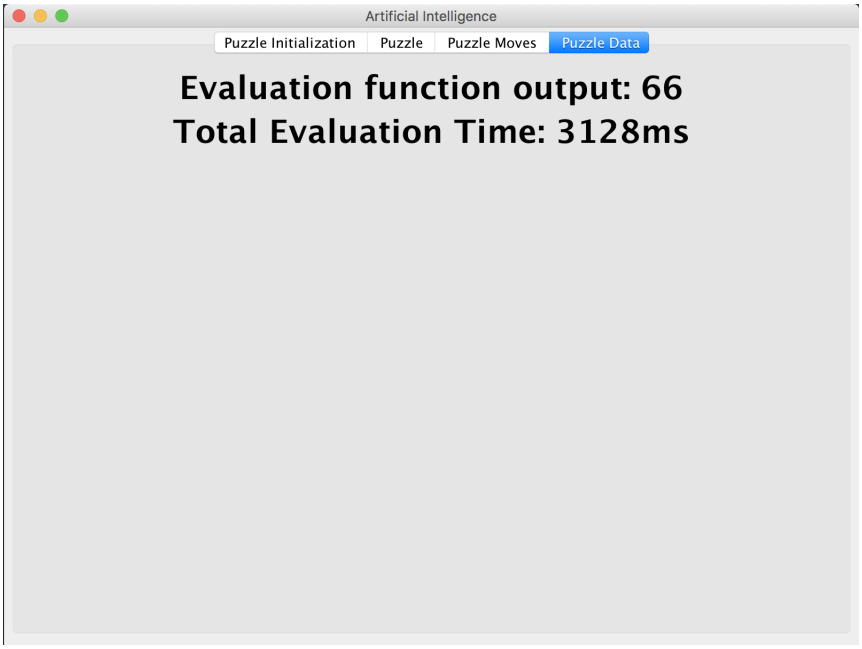

Figure 95: Simulated Annealing Best Puzzle Evaluation after 100,000 iterations for n = 9

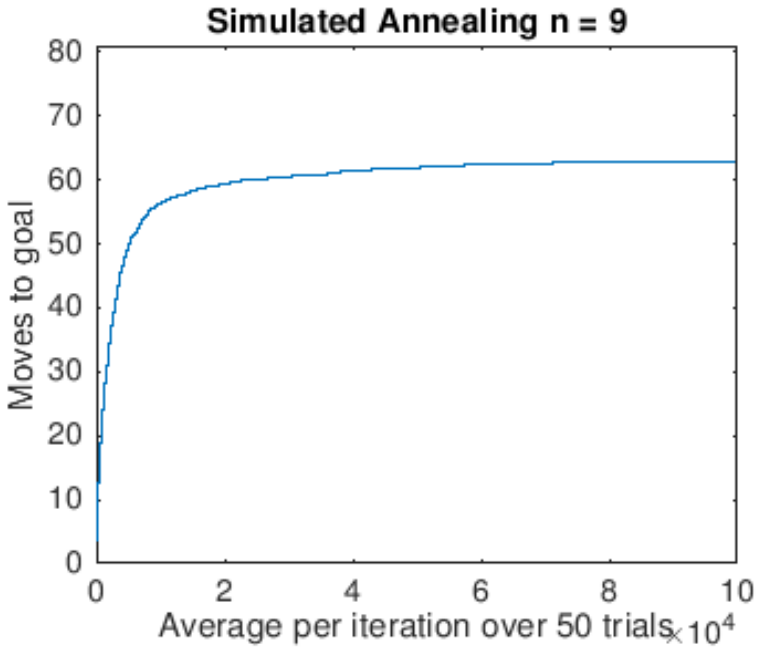

Figure 96: Plot of 100,000 iterations averaged over 50 runs for n = 9

Example Puzzle for n = 11

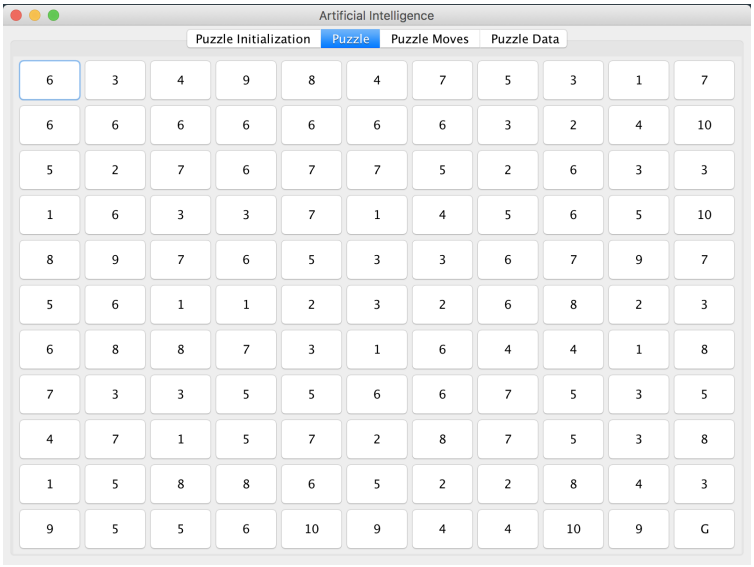

Figure 97: Simulated Annealing Best Puzzle after 100,000 iterations for n = 11

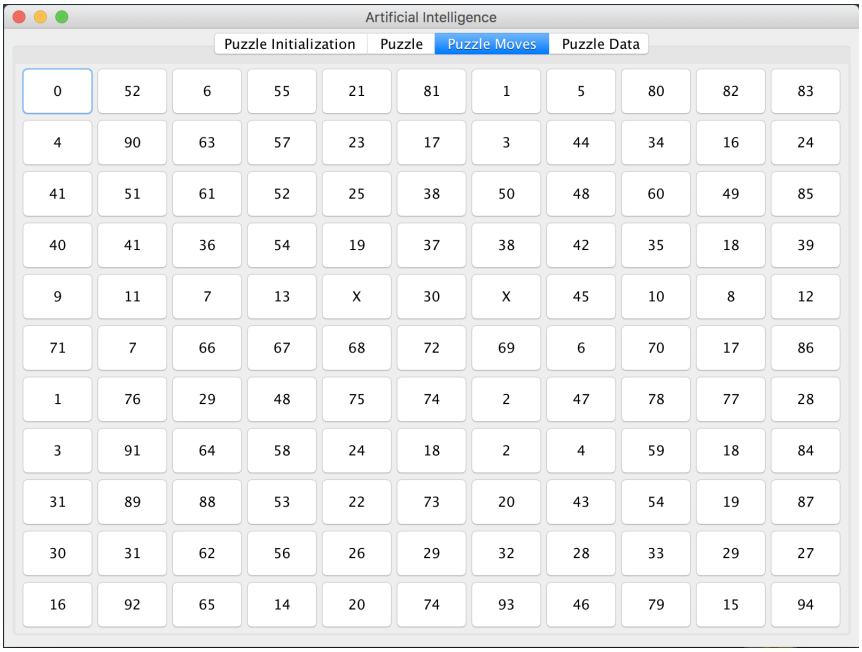

Figure 98: Simulated Annealing Best Puzzle Moves after 100,000 iterations for n = 11

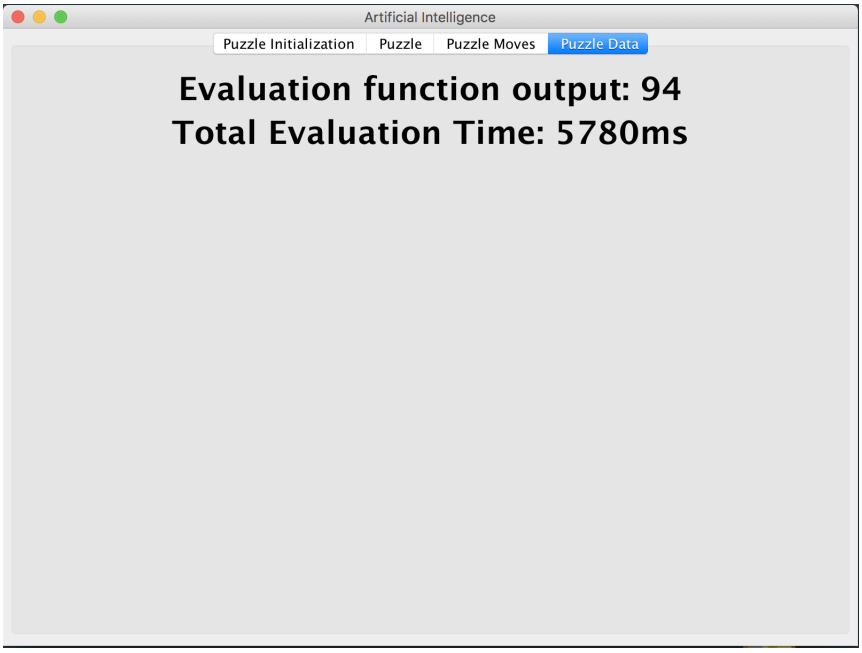

Figure 99: Simulated Annealing Best Puzzle Evaluation after 100,000 iterations for n = 11

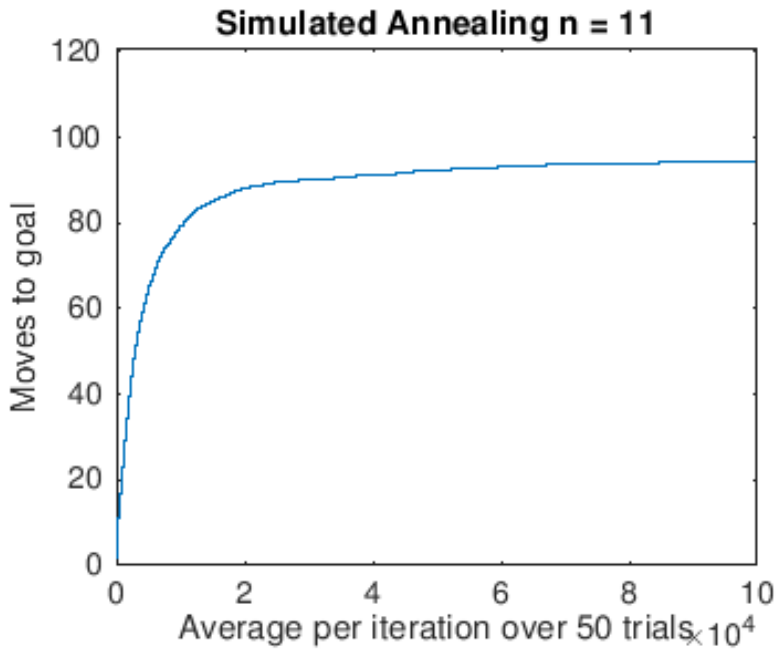

Figure 100: Plot of 100,000 iterations averaged over 50 runs for n = 11

Proposal and Implementation of a population based approach

Parameters

- Initial Population (2 - 1000) - Probability of Mutation (0 - 1.000)

Initial population determines how many random puzzles we will start out with. Per iteration, the population will double. Also no members of the population will die off (no puzzle configurations will be removed). Child puzzles will eventually breed with the parent puzzles.

Selection

The population is ranked based on their fitness, the evaluation function. Selection for breeding is done according to this ranking. The top two candidates will be bred together, and then the next two candidates will breed. So on and so forth until the last candidate.

We implement culling by removing and puzzles that have an evaluation function result less than 0.

We also remove puzzles that have a evaluation value that is less than half the current population's average.

This reduces the population size and allows more iterations to take place. Though the repurcussions of this could be that the maximum is limited to the local max.

This goes to show that the population model requires a strong population control model so that the population doesn't grow out of control with too high a population and minimal iterations.

The selection process is based on the assumption that breeding high evaluation value puzzles will lead to better evaluation values. Based on this, we use a deterministic algorithm to breed. We could have used a probabilistic approach where a higher evaluation value led to a higher probability of being bred with.

This is another approach we could have tried, however . Breeding deterministically is not in the spirit of the population approach, where progress is made on a probabalistic basis and not deterministically.

Crossover

The crossover is done by randomly selecting a point in the string representation of the puzzle. A random point is chosen, with the two selected candidates, their string will be split at this point and combined with the other puzzle. Example: aabbccdde—eff –¿ aabbccddebaa ffeeddccb—baa –¿ ffeeddccbeff

We don't have a means of distinguishing which parts of a puzzle are high in fitness, so this is why the crossover point is randomly chosen.

When the children are created, they are entered into the population. The parents are still in the population, and are not removed when children are created.

Mutation

For the mutation, all chromosomes in the candidate (all characters in a string) have a chance of being replaced with a new random value. This value is of course chosen at random, and also so the value leads to a valid move in the puzzle.

The probability of the mutation of each chromosome is chosen by the user.

Output

- fitness of highest fitness candidate. The candidate will have the highest fitness in the population. This is done by ensuring that the population is stored in a

Technical details

The algorithm was implemented with the help of a priority queue, to implement a priority heap. When the population is created, they are sorted in a priority heap to rank them based on their fitness function (puzzle's evaluation function). When children are bred through the crossover process, the children are added into the population by insertion through the priority heap. This ensures that the population is ranked and the heap is ordered to prepare for the next crossover. This also helps easily determine the highest fitness candidate.

Figures

For: mutation = 0.05 initial population = 60

Example Puzzle for n = 5

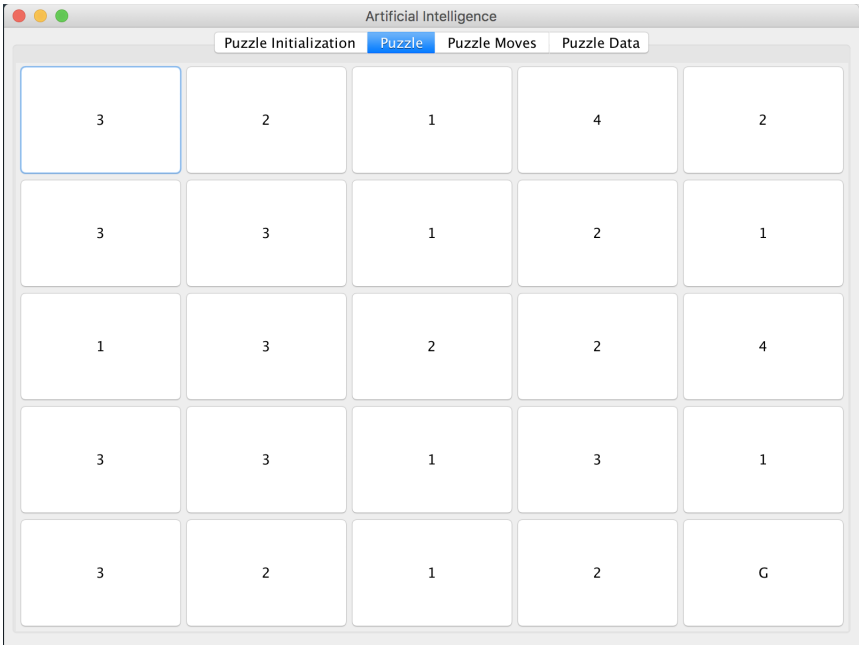

Figure 101: Population Approach Puzzle after being run for 620ms on puzzle of size n = 5

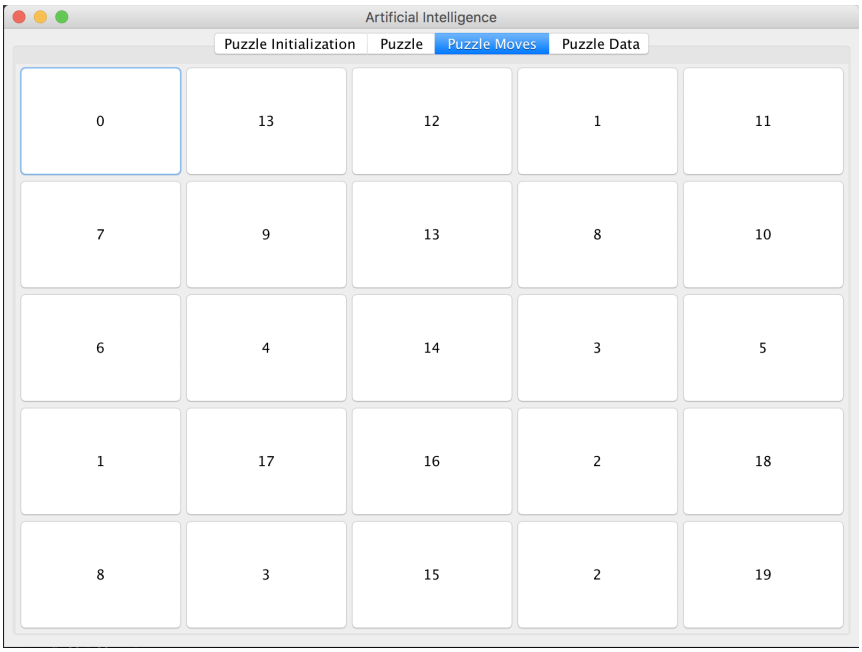

Figure 102: Population Approach Puzzle Moves after being run for 620ms on puzzle of size n = 5

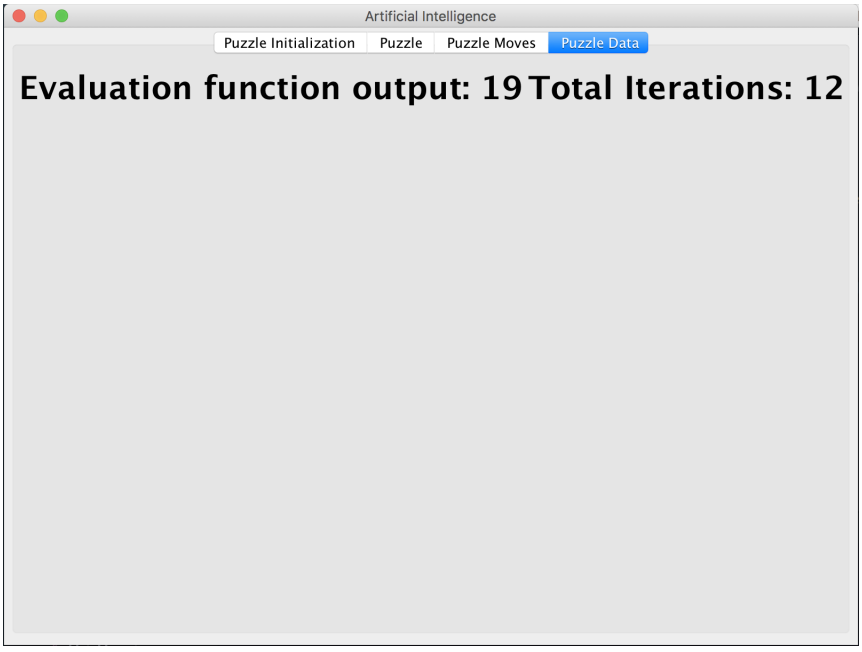

Figure 103: Population Approach Puzzle Data after being run for 620ms on puzzle of size n = 5

Example Puzzle for n = 7

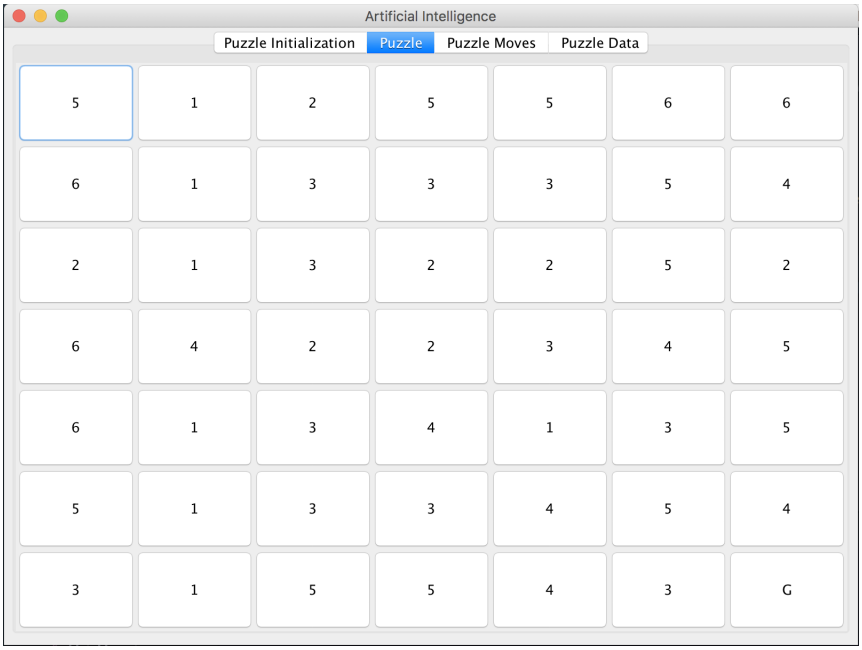

Figure 104: Population Approach Puzzle after being run for 1420ms on puzzle of size n = 7

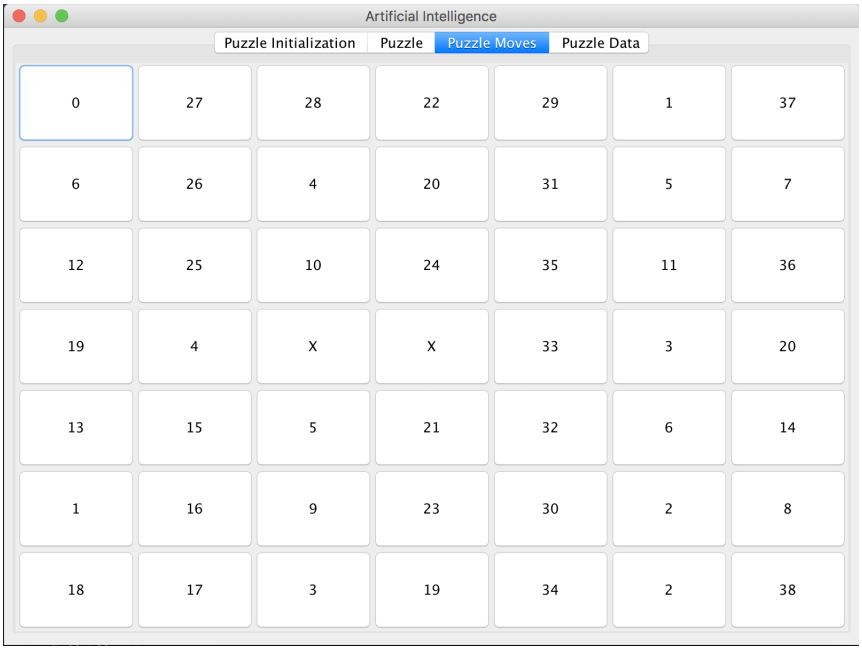

Figure 105: Population Approach Puzzle Moves after being run for 1420ms on puzzle of size n = 7

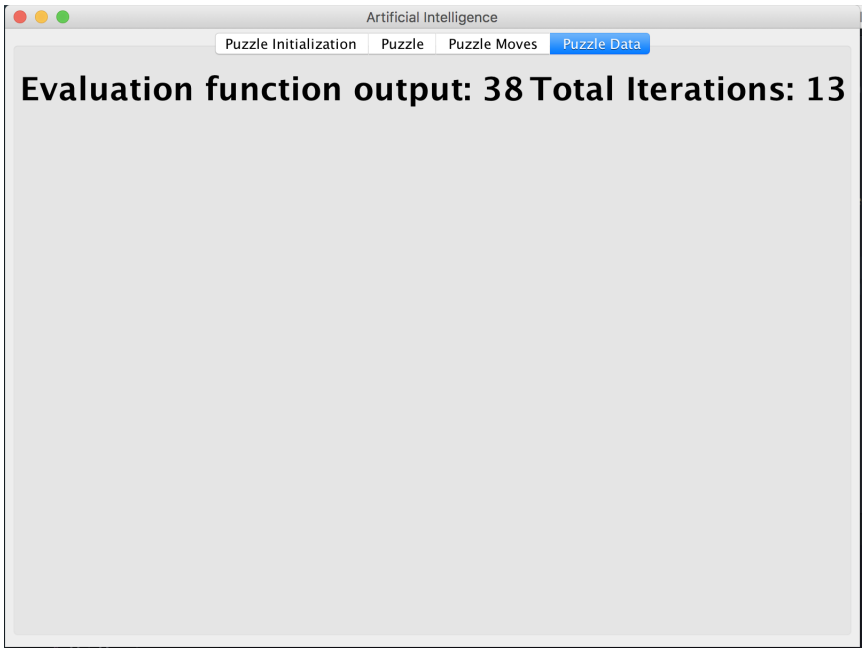

Figure 106: Population Approach Puzzle Data after being run for 620ms on puzzle of size n = 7

Example Puzzle for n = 9

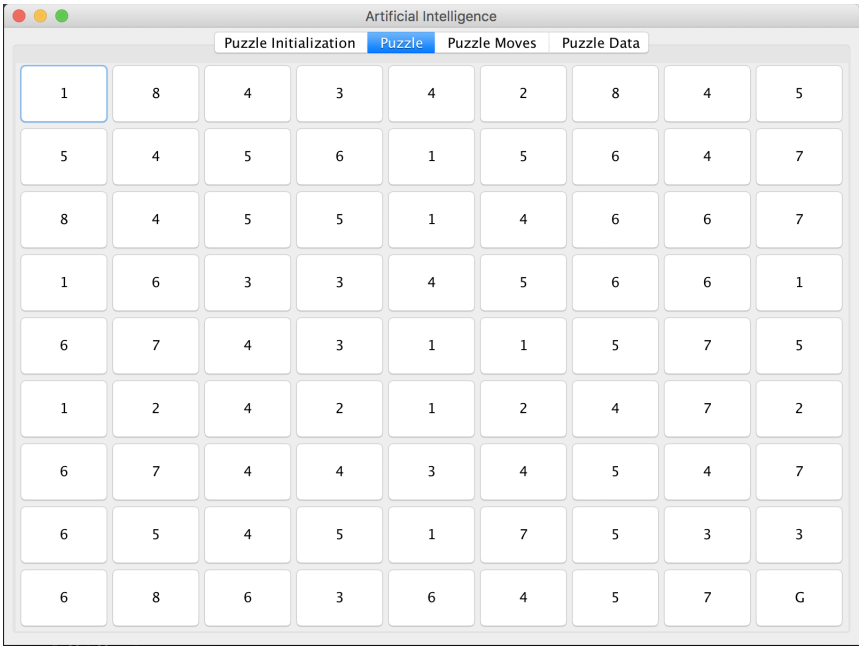

Figure 107: Population Approach Puzzle after being run for 3120ms on puzzle of size n = 9

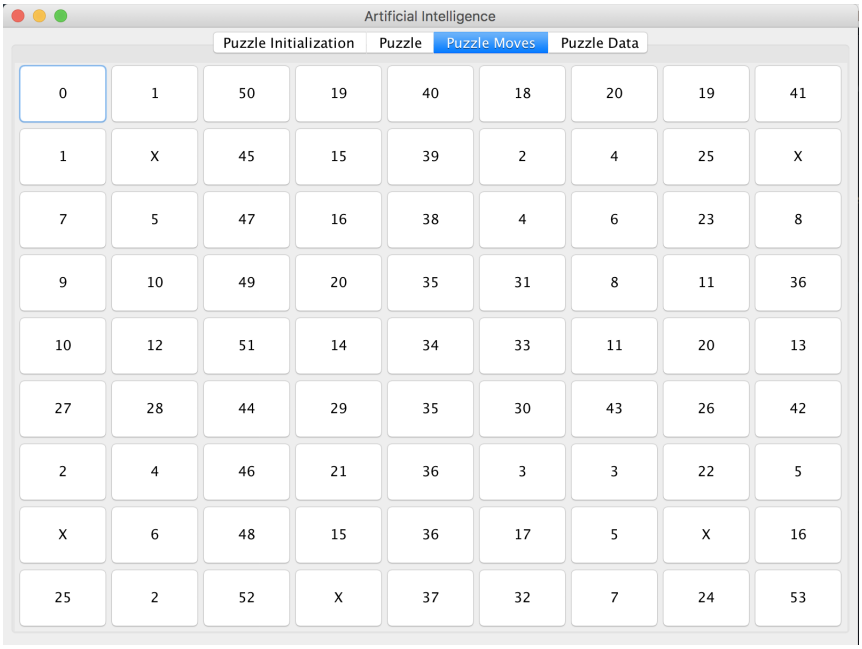

Figure 108: Population Approach Puzzle Moves after being run for 3120ms on puzzle of size n = 9

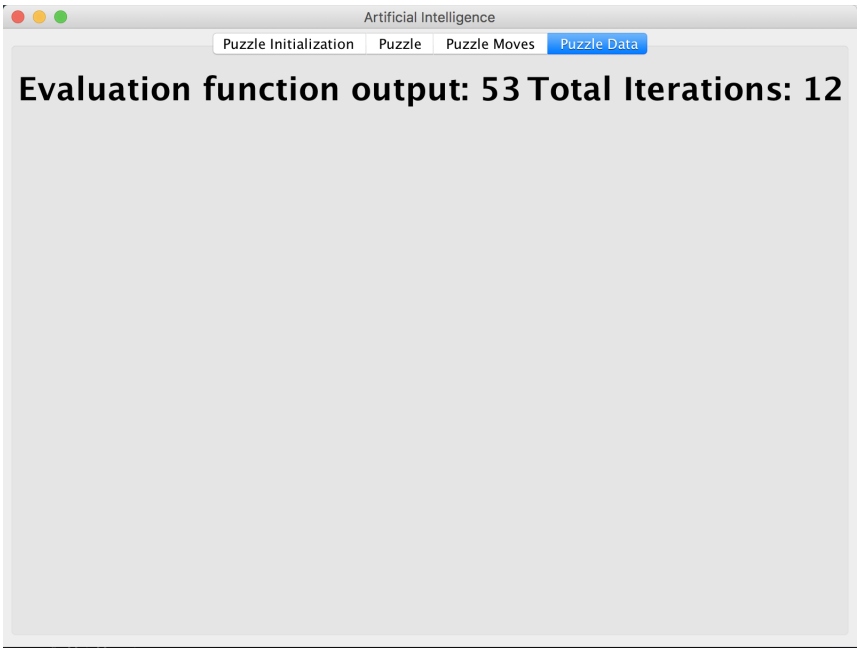

Figure 109: Population Approach Puzzle Data after being run for 3120ms on puzzle of size n = 9

Example Puzzle for n = 11

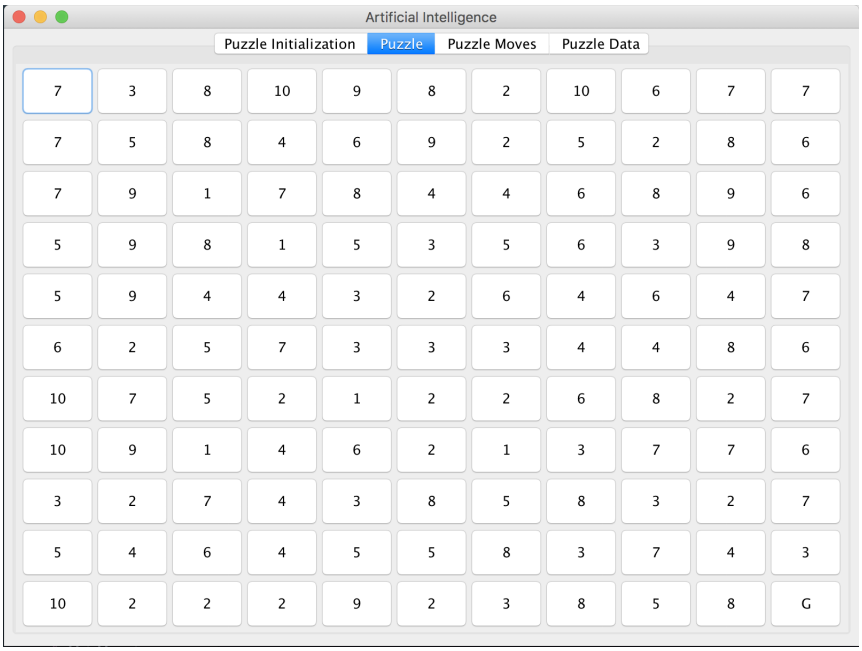

Figure 110: Population Approach Puzzle after being run for 5780ms on puzzle of size n = 11

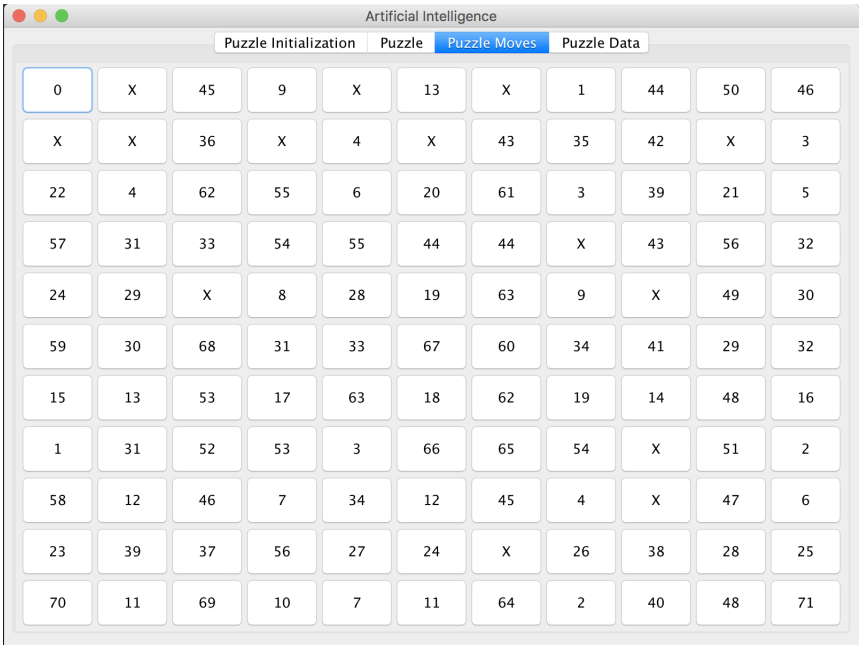

Figure 111: Population Approach Puzzle Moves after being run for 5780ms on puzzle of size n = 11

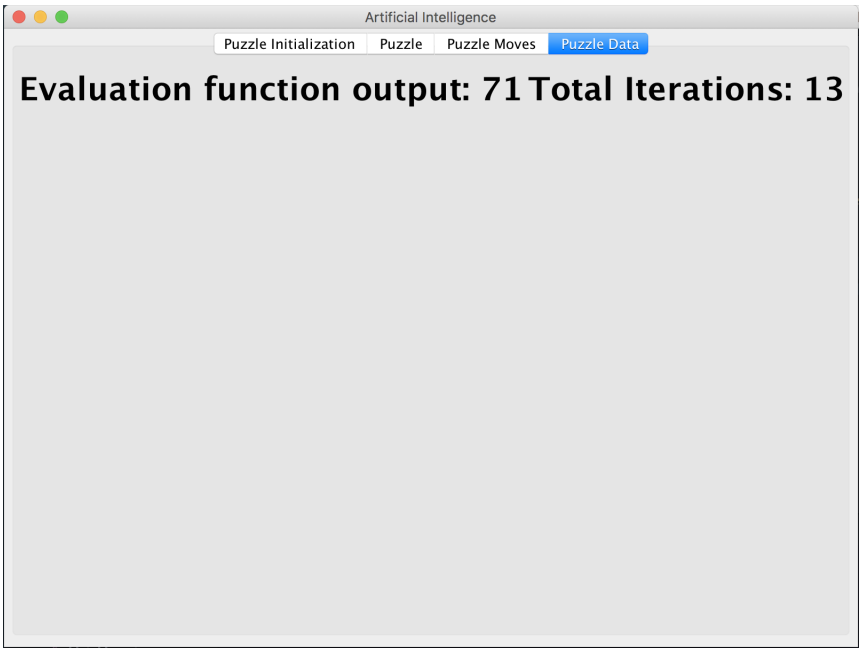

Figure 112: Population Approach Puzzle Data after being run for 5780ms on puzzle of size n = 11