AI-Intro Assignment 3 - Using the A* Algorithm

Tom Meland Pedersen

October 3, 2014

1 Pathfinding in 2-D games

The following images where generated using the c++ program in the source folder. The boards where given as input to the the command line call of the <algorithm>.exe file. In order to compile different algorithms and/or turn off/on open and closed node marks, change the preprossessor macros ALG and NODEMARKS in boardFunctions.h. Several other settings may be specified by changing the constants in astarMain.h and boardFunctions.h.

Example call to the A* algorithm: ./astar boards/board-1-4.txt runs the A* algorithm on board 1.4.

The process may be animated by giving an optional 3'rd argument. For example: ./astar boards/board-1-4.txt arg3

SDL2 was used to create the board graphics.

1.1 Grids with obstacles

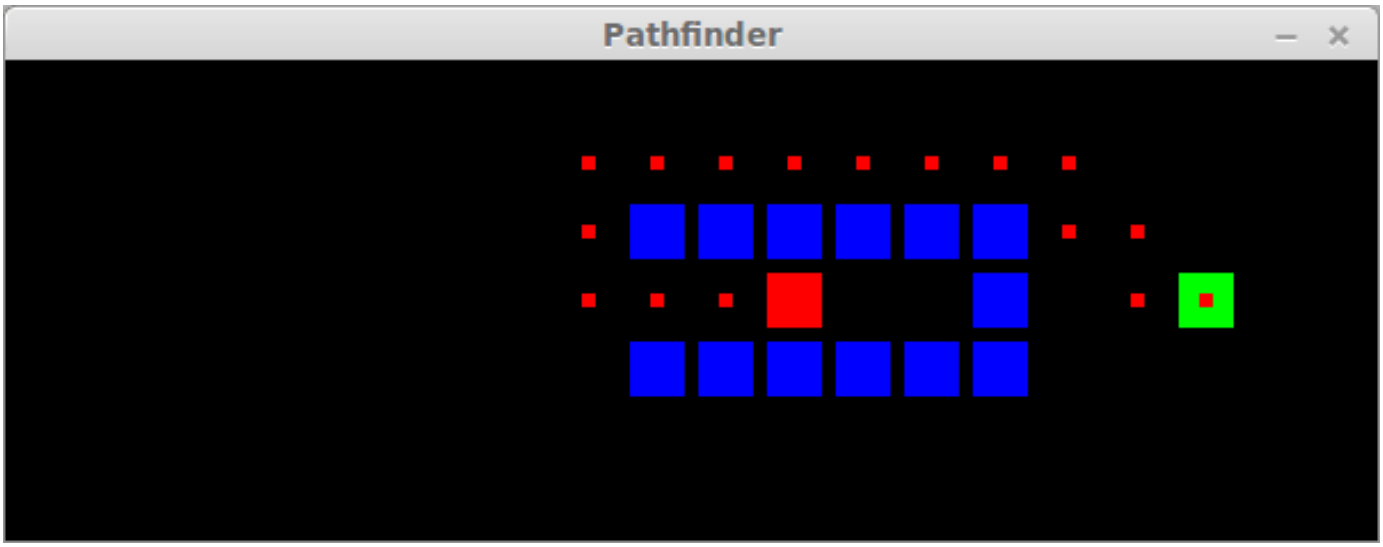

Figure 1: Shortest path using A* for board 1.1.

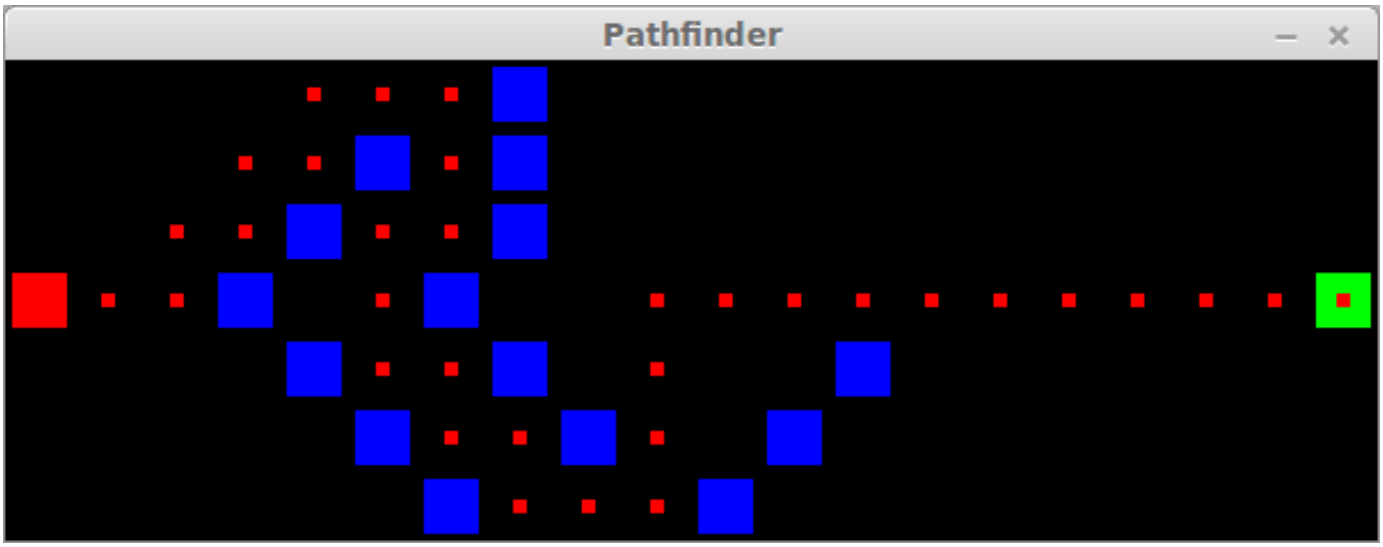

Figure 2: Shortest path using A* for board 1.2.

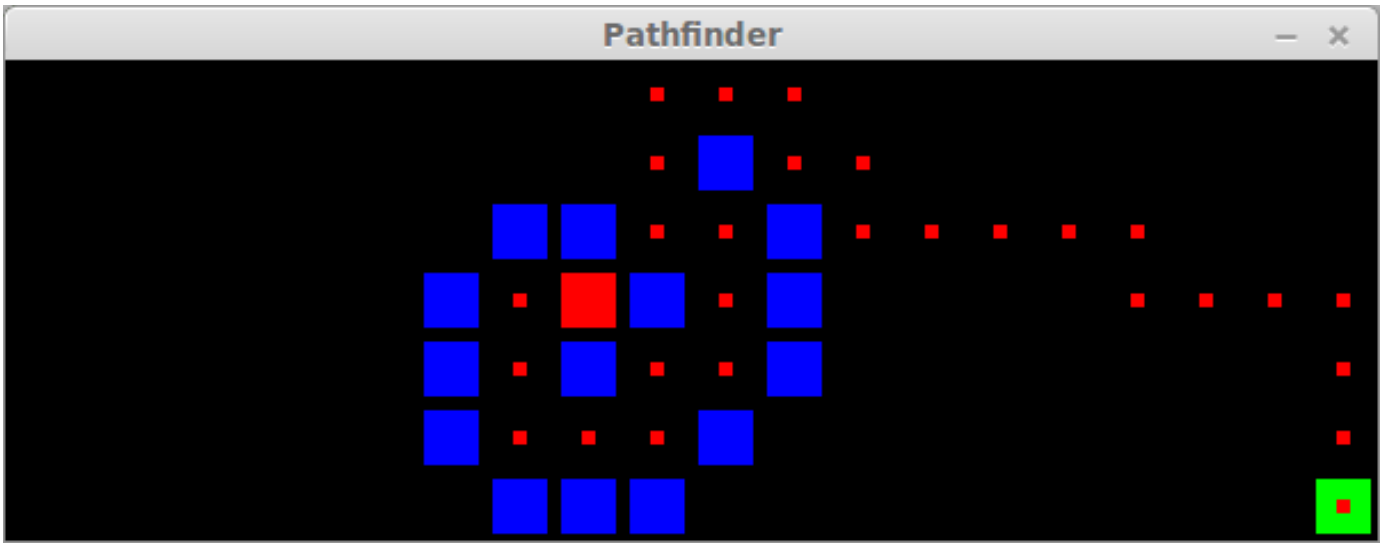

Figure 3: Shortest path using A* for board 1.3.

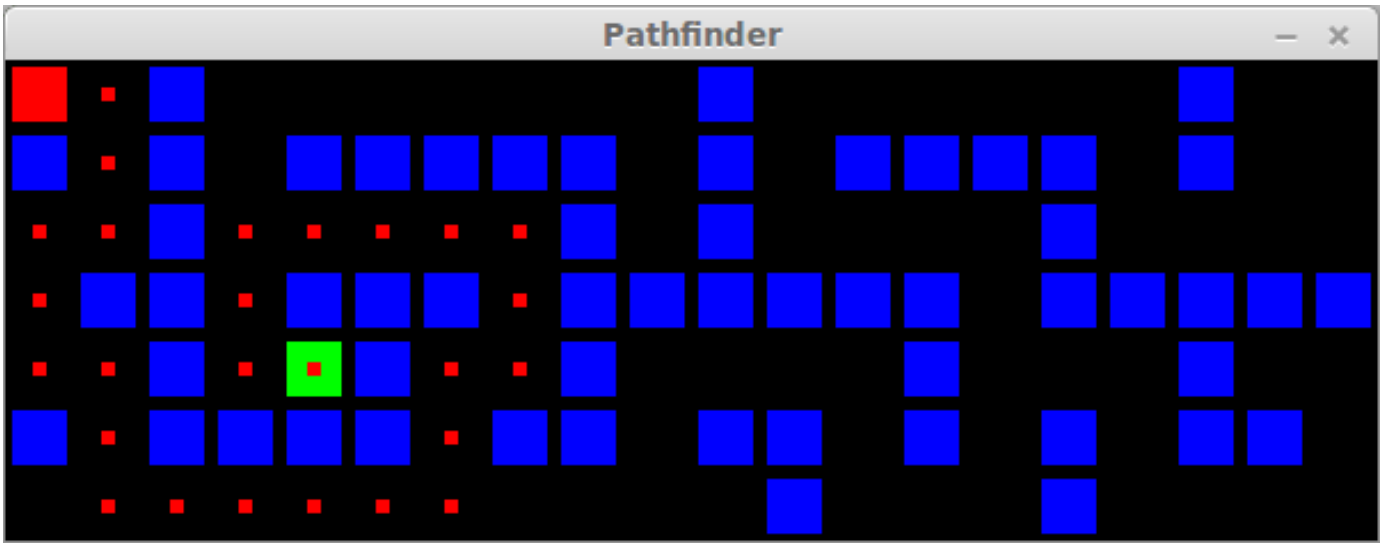

Figure 4: Shortest path using A* for board 1.4.

1.2 Grids with different cell costs

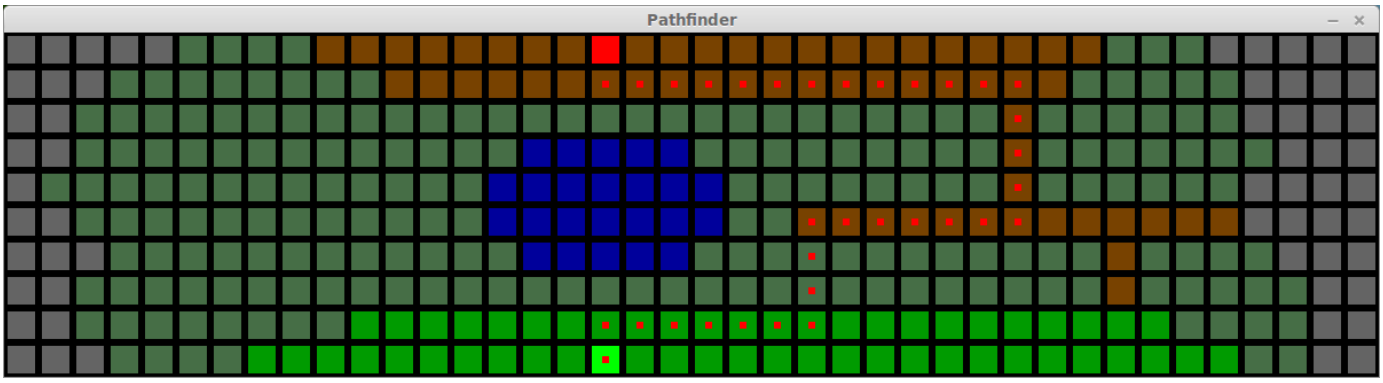

Figure 5: Shortest path using A* for board 2.1.

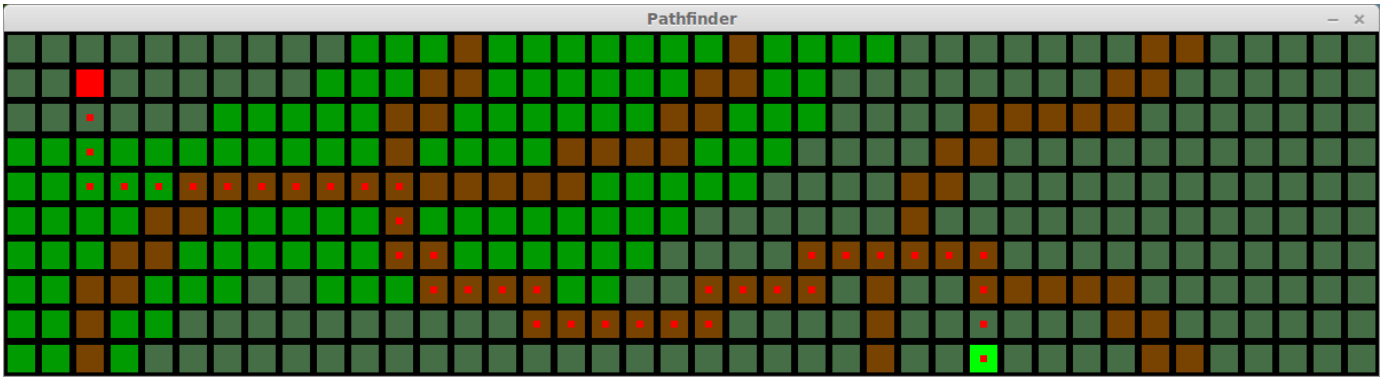

Figure 6: Shortest path using A* for board 2.2.

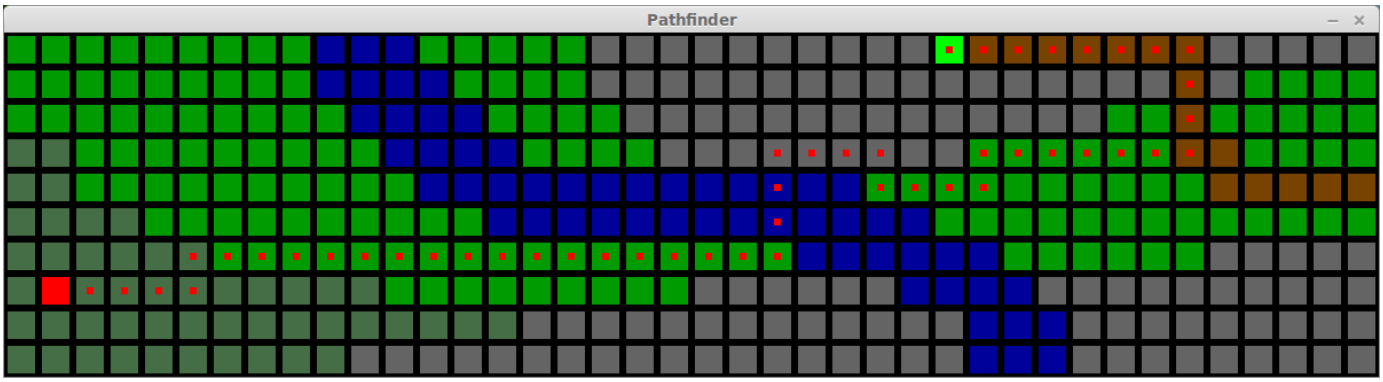

Figure 7: Shortest path using A* for board 2.3.

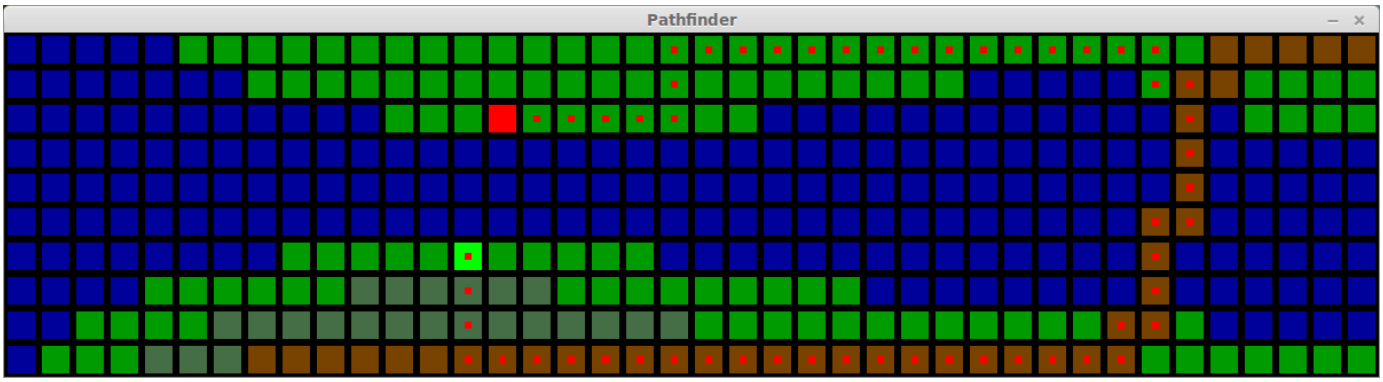

Figure 8: Shortest path using A* for board 2.4.

1.3 Comparison with BFS and Dijkstra's Algorithm

Open nodes are marked with white dots, closed nodes are marked with cyan crosses.

In the following section, better performance refers to less nodes examined before reaching the goal.

One can clearly see in the following images that bfs ignores tile costs and finds the shortest possible manhattan distance from the source to the goal. It is also inefficient (examines more nodes) compared to the two others as it examines all nodes around the source and expands outwards, disregarding node cost and direction.

Dijkstras algorithm performs better in most cases as it takes node cost into account, however, if all nodes have the same cost it will perform almost as bad as bfs as it does not take direction into account.

The A* algorithm, takes into account direction as well as node cost by adding the distance estimate. Thus it will tend to move towards the goal unless it becomes very costly, in wich case it looks for alternatives. It especially outclasses dijkstras algorithm when all nodes are of the same/similar cost or in clusters where they have similir costs locally.

In many cases the shortest paths found will vary between the algorithms. This is due to the fact that they process nodes in different order. In bfs it depends on what node gets added first as this node will be examined before the ones following regardless of node cost. In dijkstra, the heap does not take estimated distance into account when sorting the nodes, as such, two nodes that have different cost in the A* case might have the same cost in the dijkstra case, hence it will process the ones with the "same" cost in a FIFO manner (similar to bfs) where A* would prioritize the ones closest to the goal.

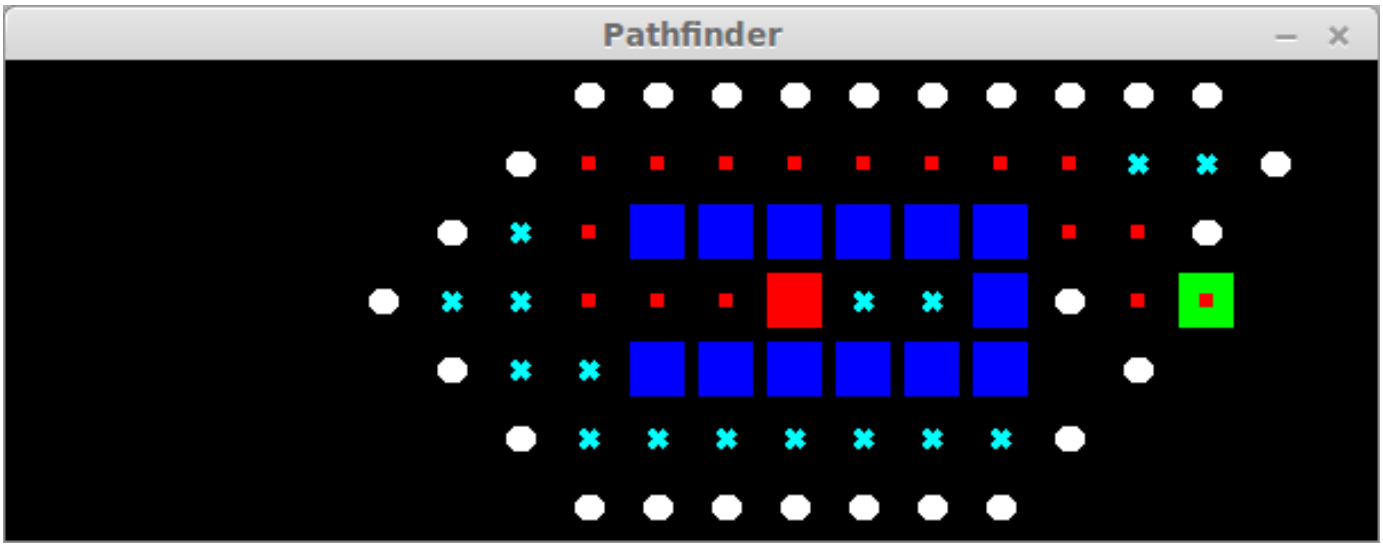

Figure 9: Shortest path using A* with closed and open nodes marked for board 1.1.

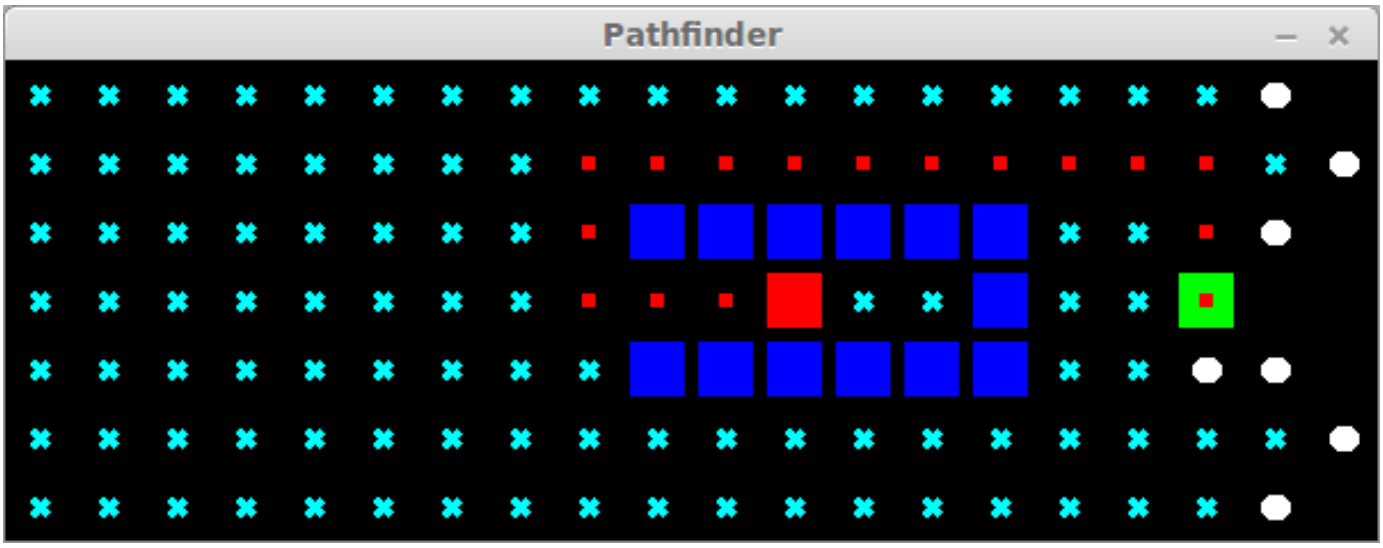

Figure 10: Shortest path using dijkstra with closed and open nodes marked for board 1.1.

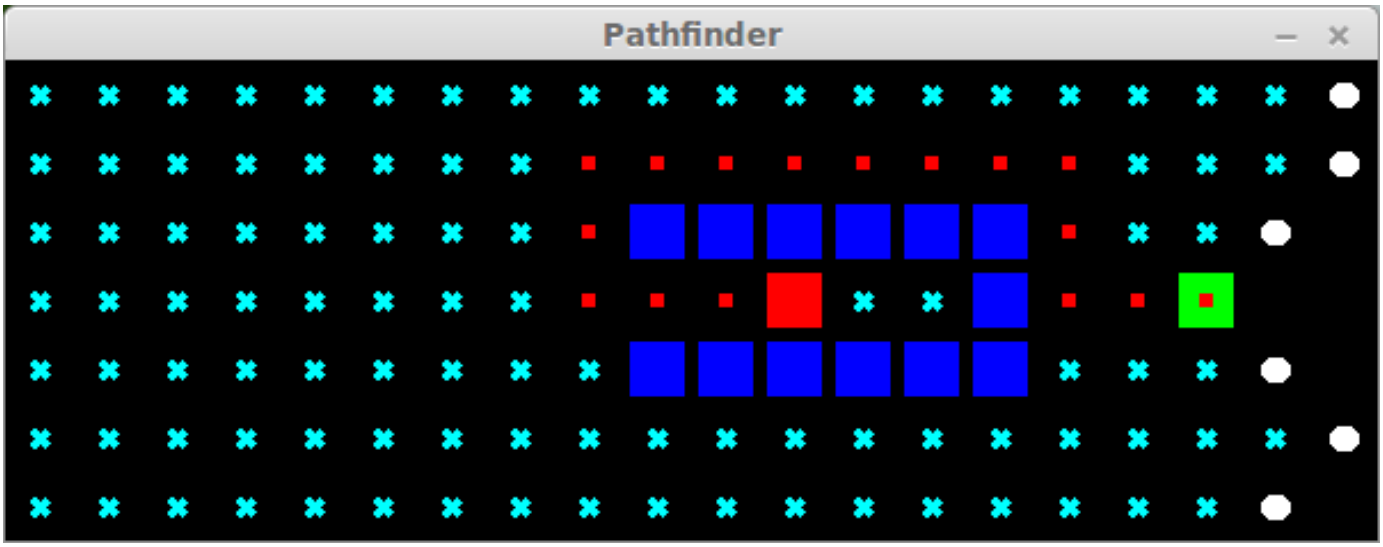

Figure 11: Shortest path using bfs with closed and open nodes marked for board 1.1.

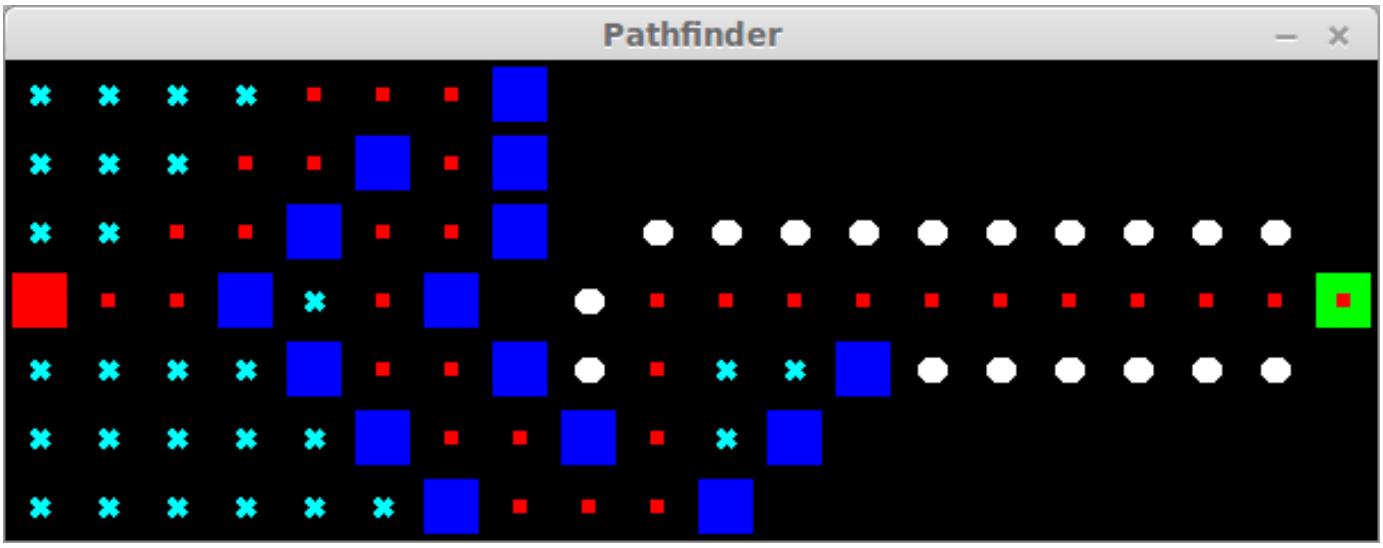

Figure 12: Shortest path using A* with closed and open nodes marked for board 1.2.

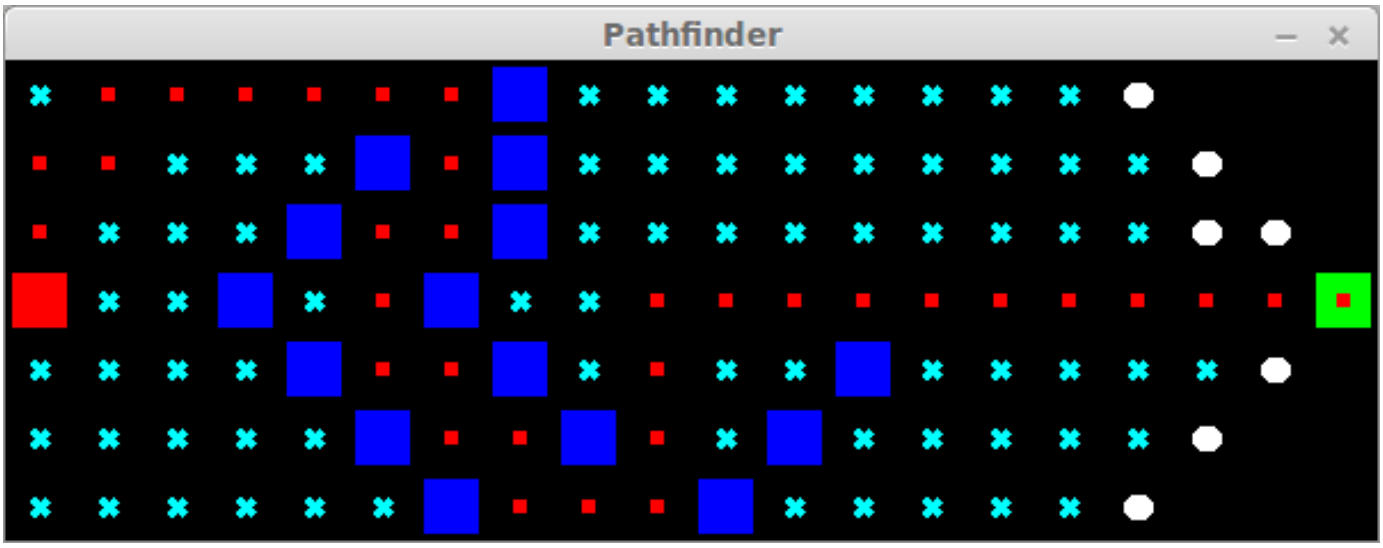

Figure 13: Shortest path using dijkstra with closed and open nodes marked for board 1.2.

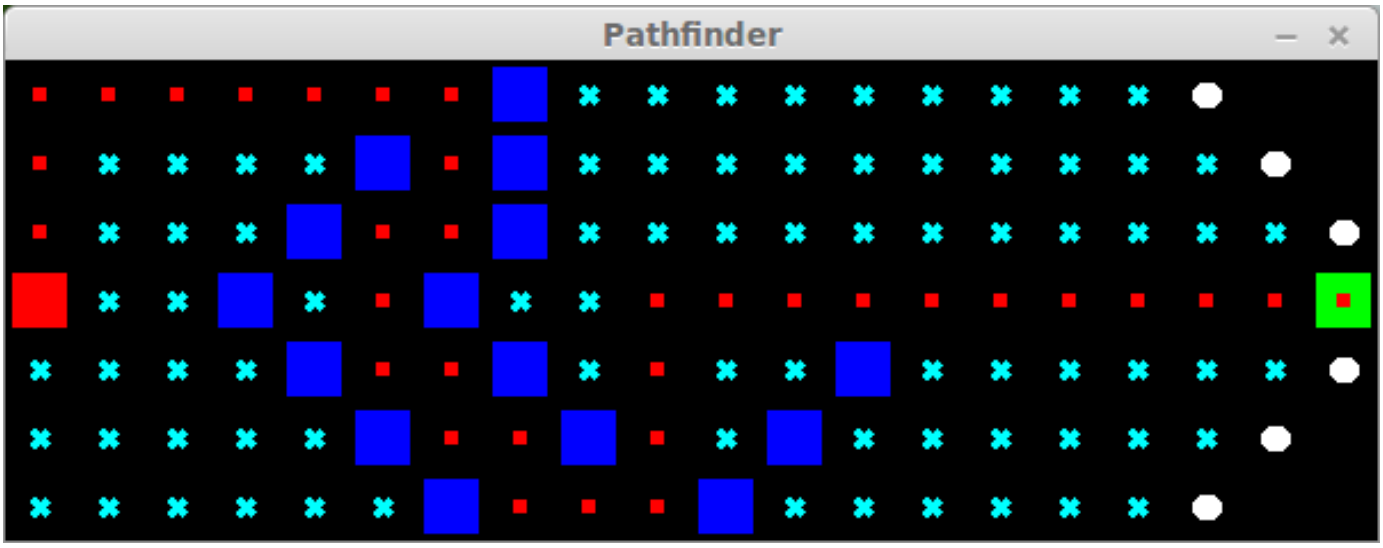

Figure 14: Shortest path using bfs with closed and open nodes marked for board 1.2.

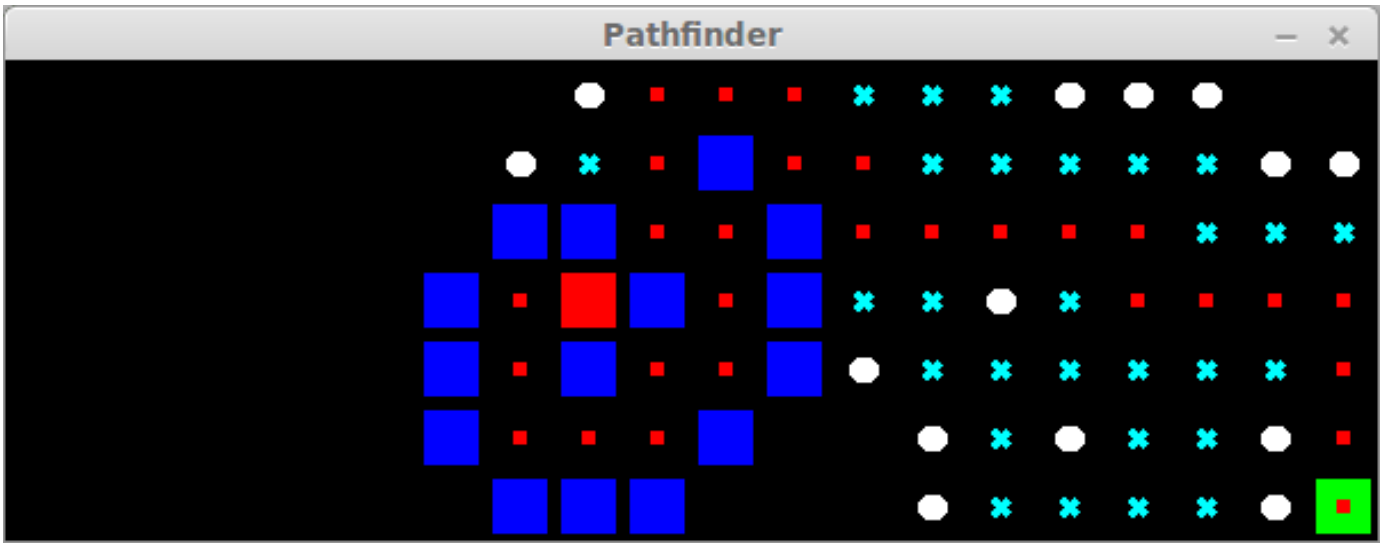

Figure 15: Shortest path using A* with closed and open nodes marked for board 1.3.

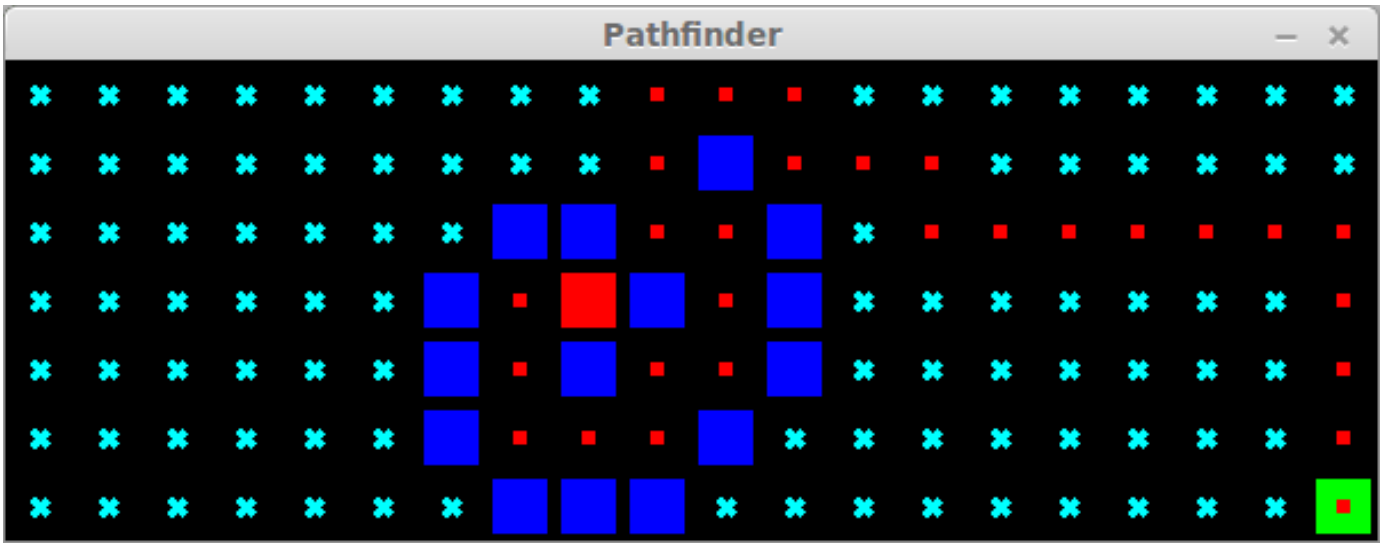

Figure 16: Shortest path using dijkstra with closed and open nodes marked for board 1.3.

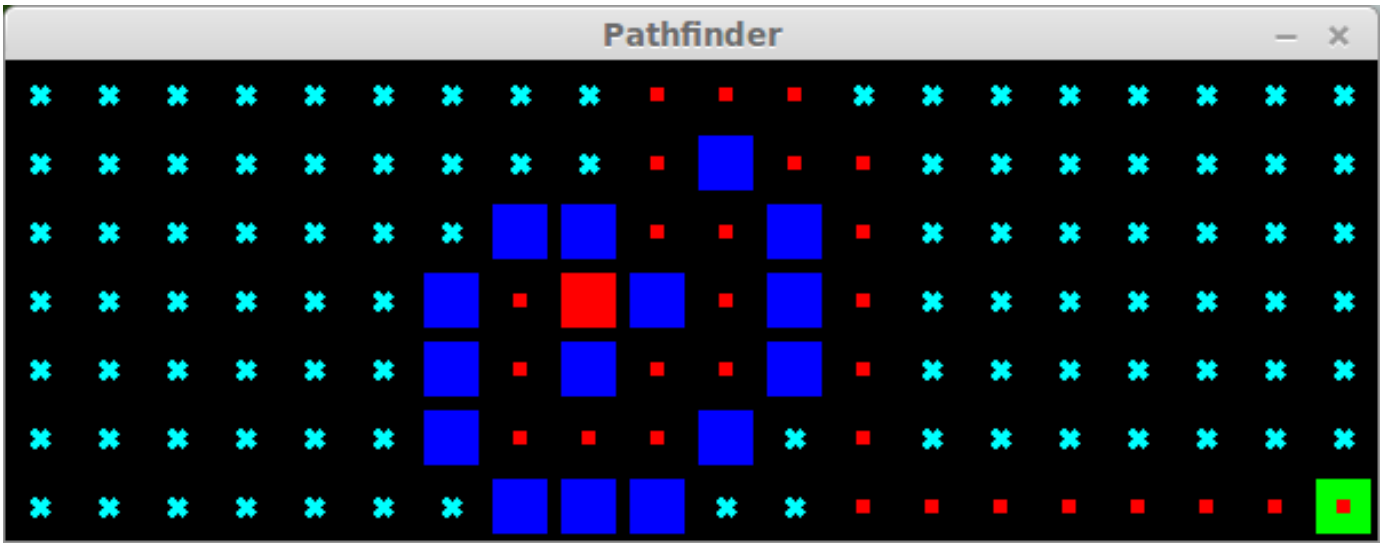

Figure 17: Shortest path using bfs with closed and open nodes marked for board 1.3.

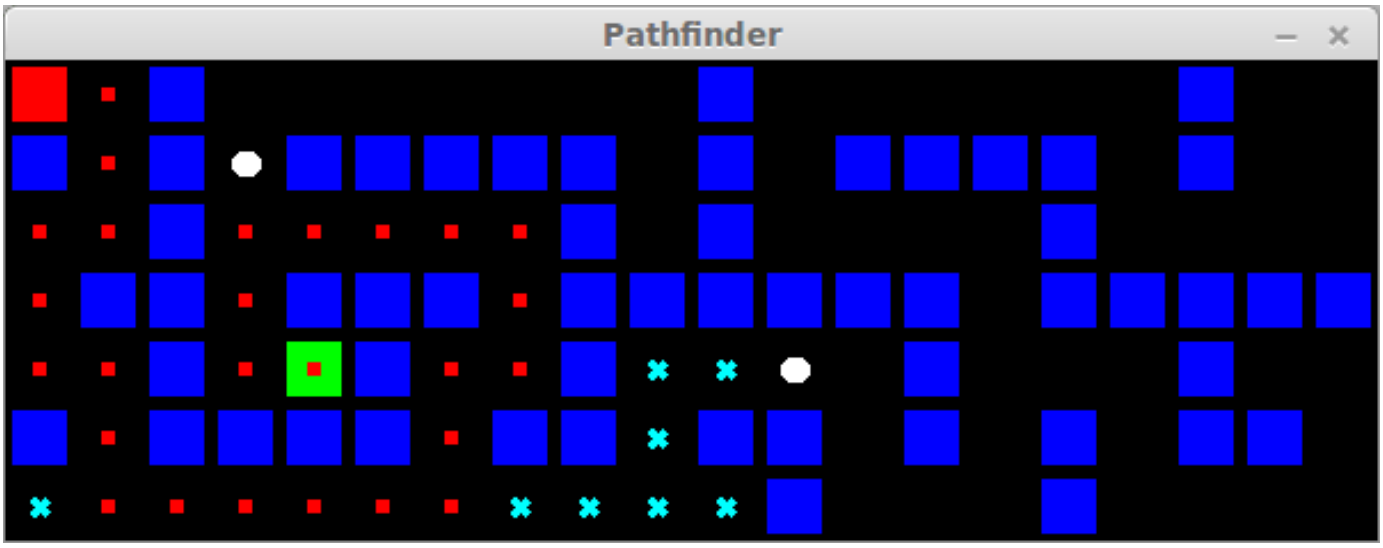

Figure 18: Shortest path using A* with closed and open nodes marked for board 1.4.

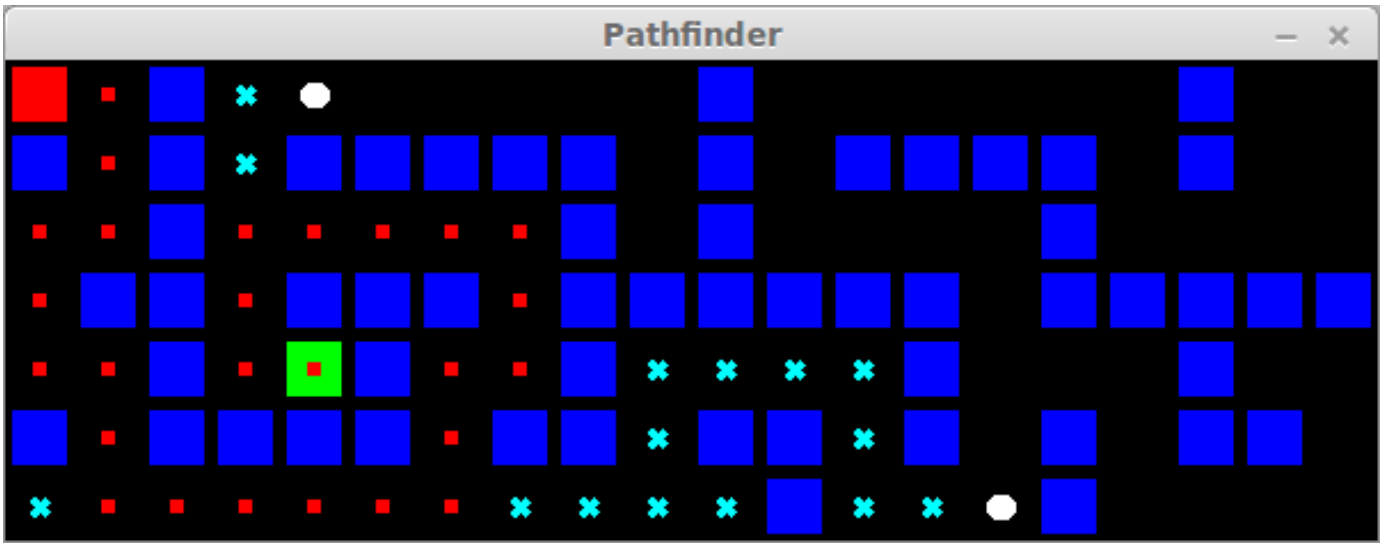

Figure 19: Shortest path using dijkstra with closed and open nodes marked for board 1.4.

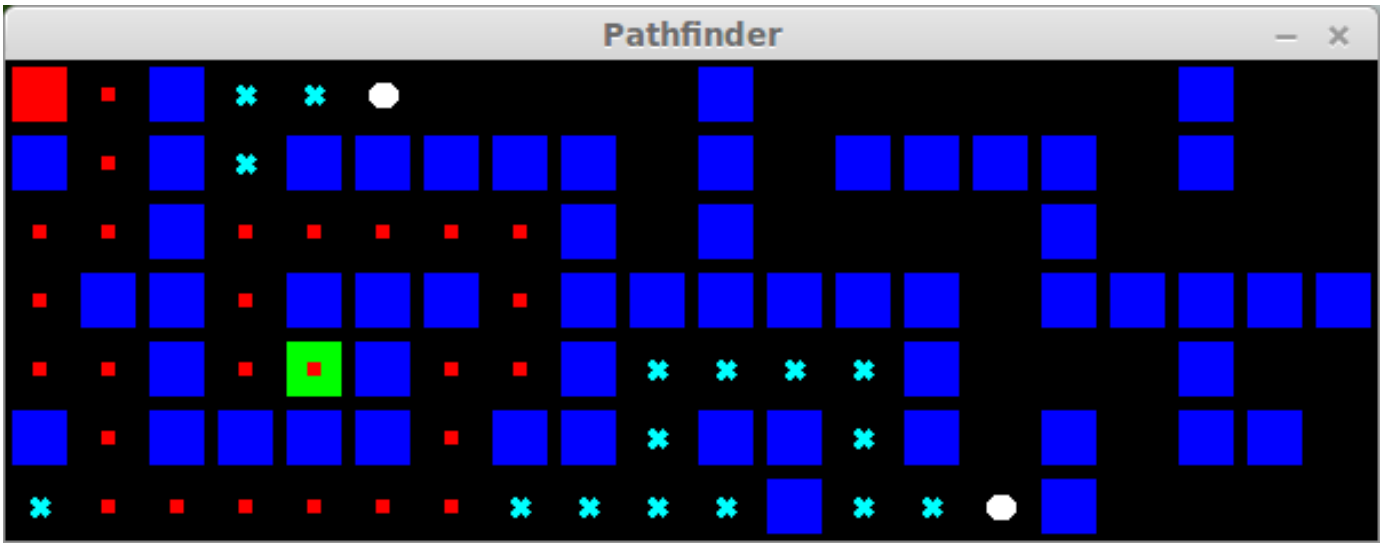

Figure 20: Shortest path using bfs with closed and open nodes marked for board 1.4.

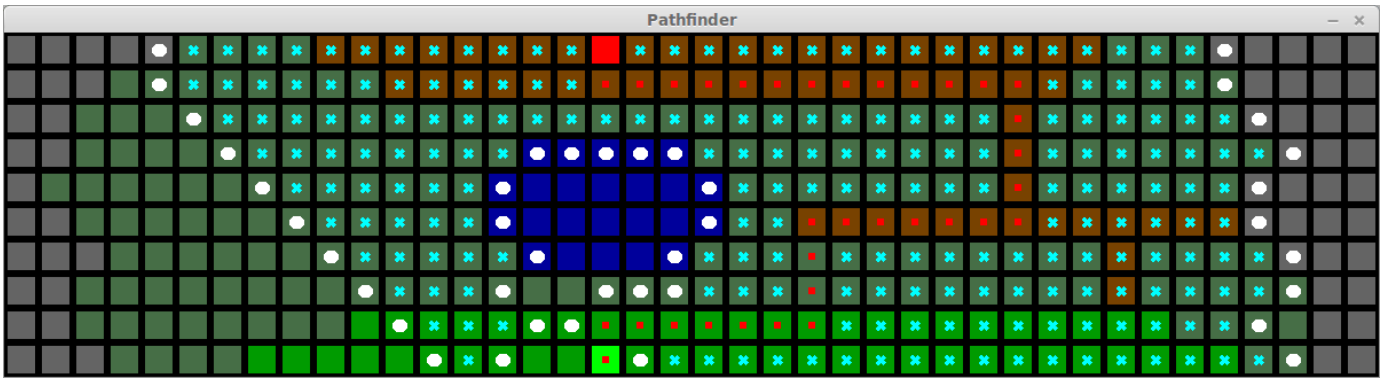

Figure 21: Shortest path using A* with closed and open nodes marked for board 2.1.

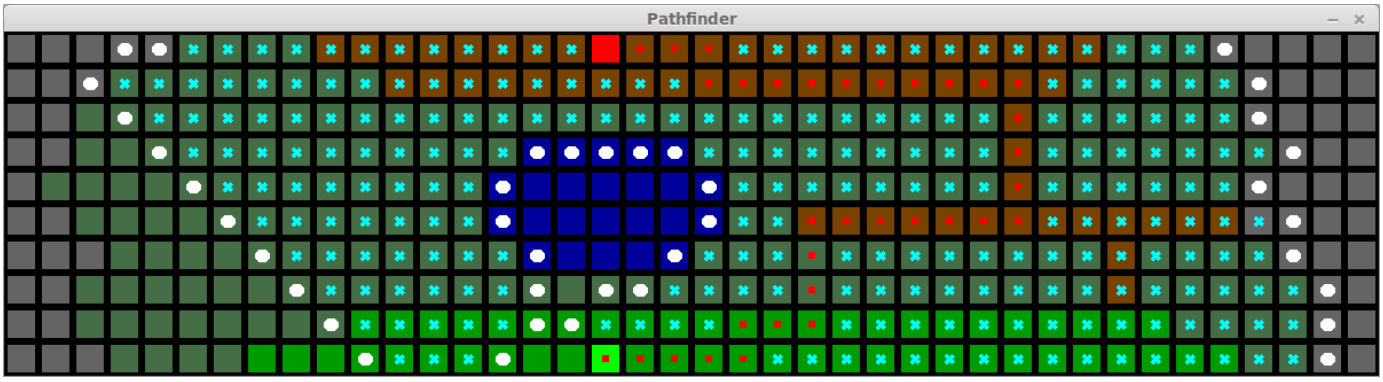

Figure 22: Shortest path using dijkstra with closed and open nodes marked for board 2.1.

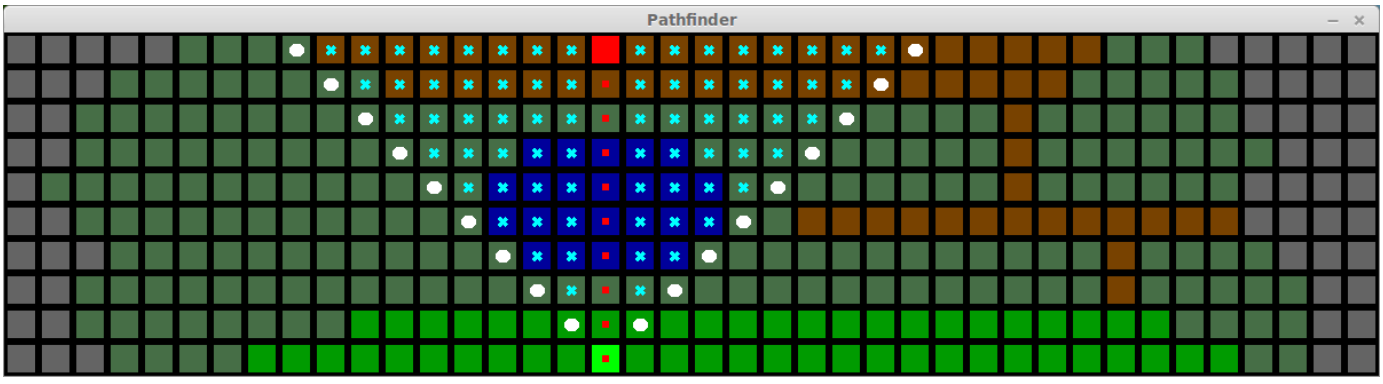

Figure 23: Shortest path using bfs with closed and open nodes marked for board 2.1.

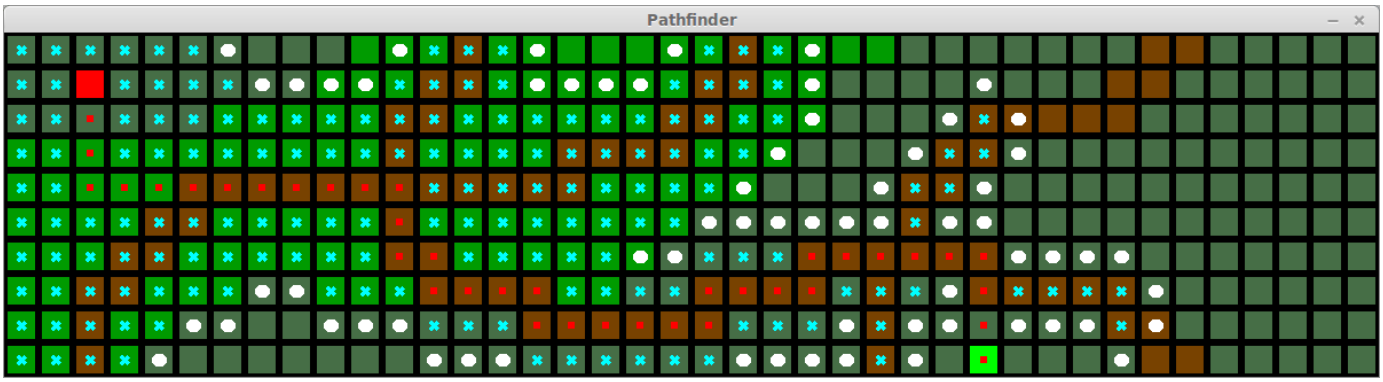

Figure 24: Shortest path using A* with closed and open nodes marked for board 2.2.

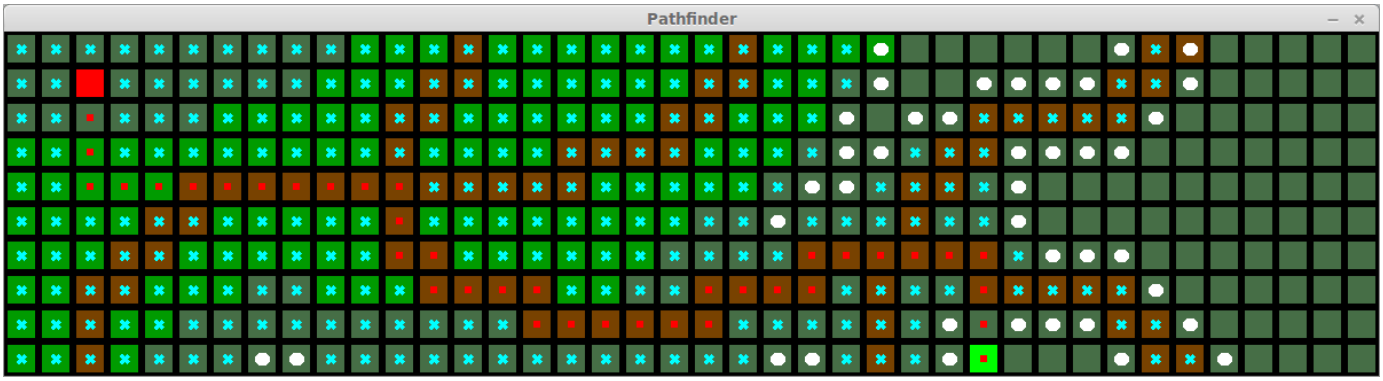

Figure 25: Shortest path using dijkstra with closed and open nodes marked for board 2.2.

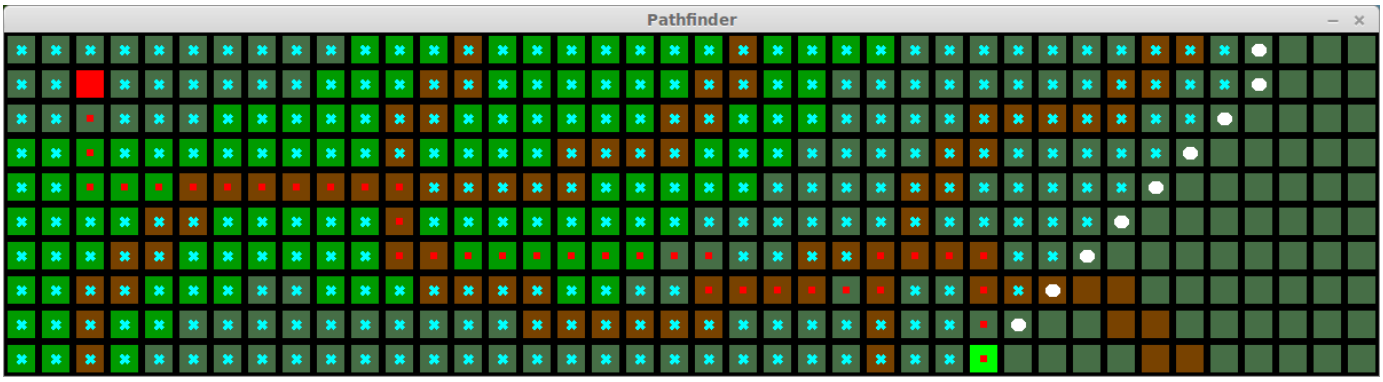

Figure 26: Shortest path using bfs with closed and open nodes marked for board 2.2.

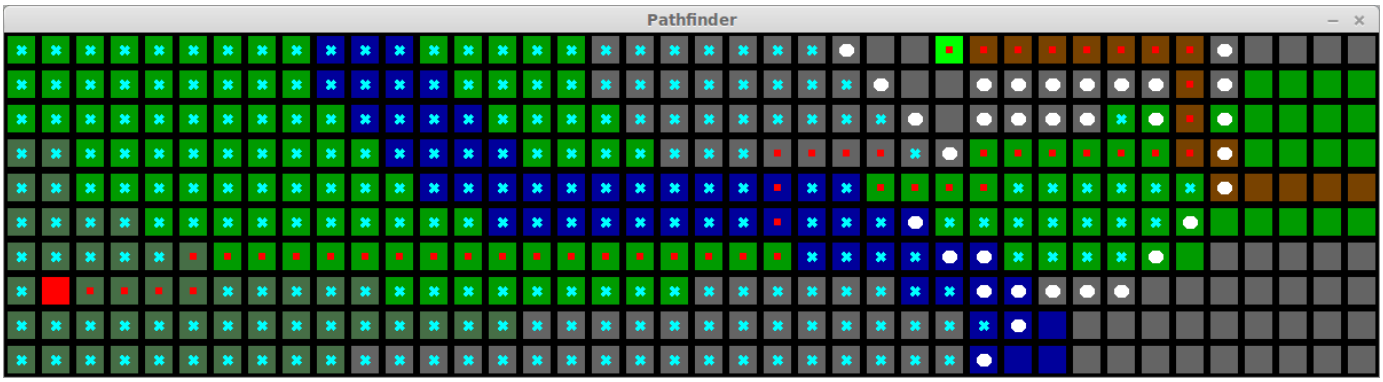

Figure 27: Shortest path using A* with closed and open nodes marked for board 2.3.

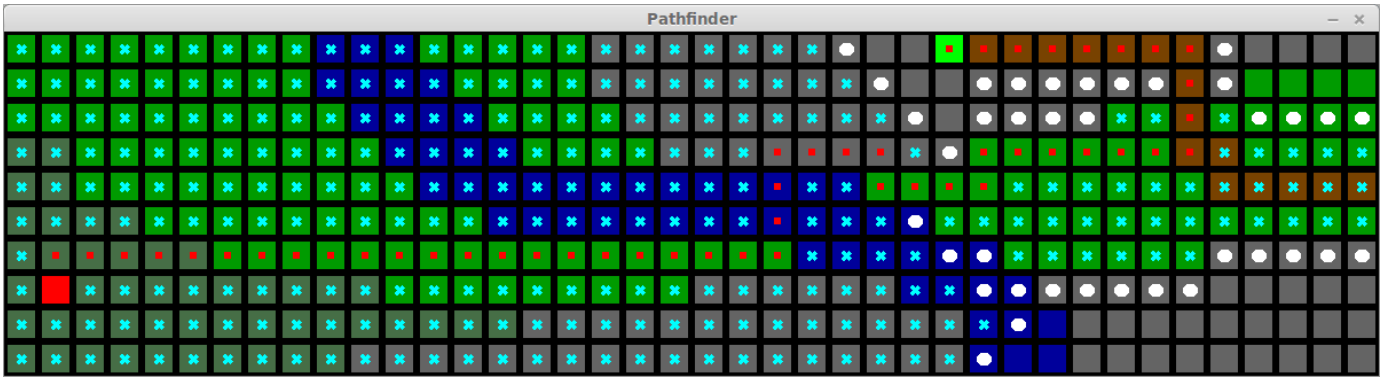

Figure 28: Shortest path using dijkstra with closed and open nodes marked for board 2.3.

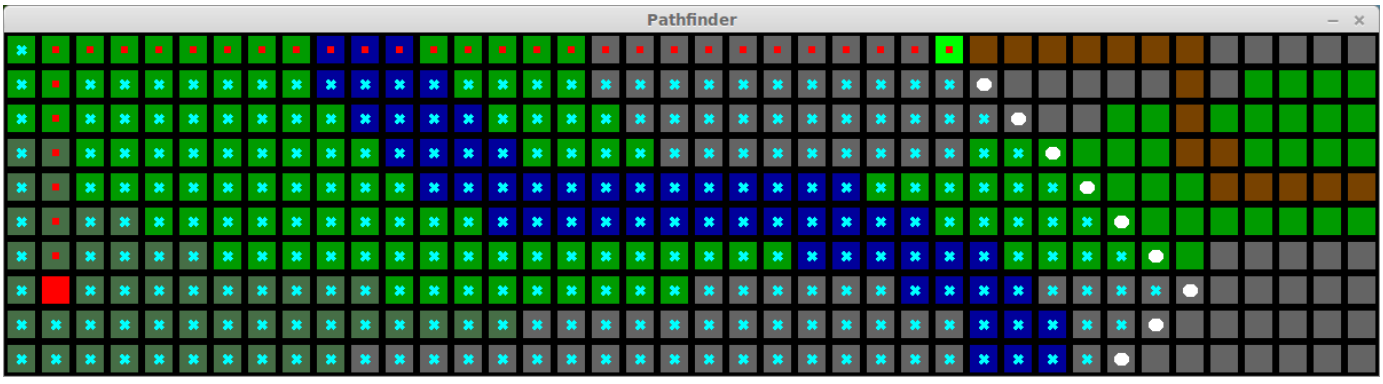

Figure 29: Shortest path using bfs with closed and open nodes marked for board 2.3.

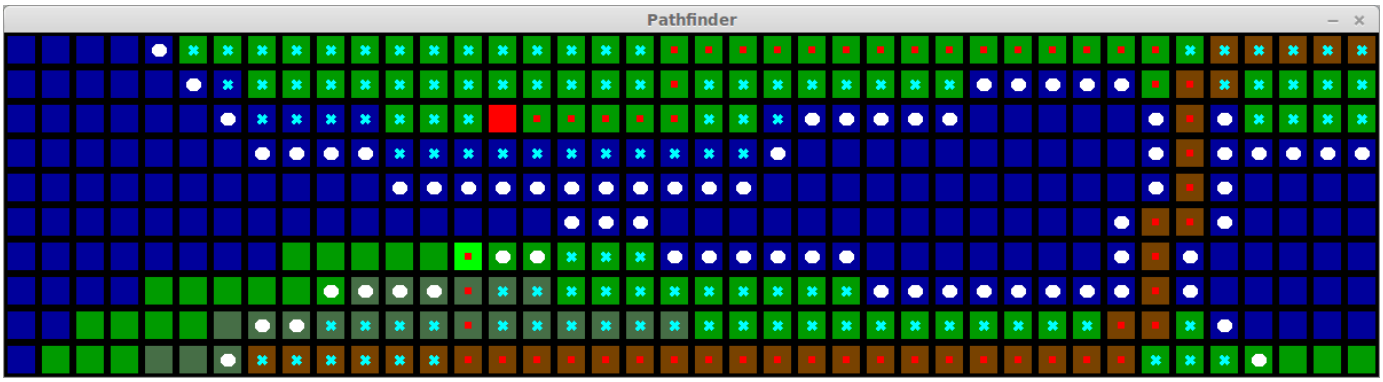

Figure 30: Shortest path using A* with closed and open nodes marked for board 2.4.

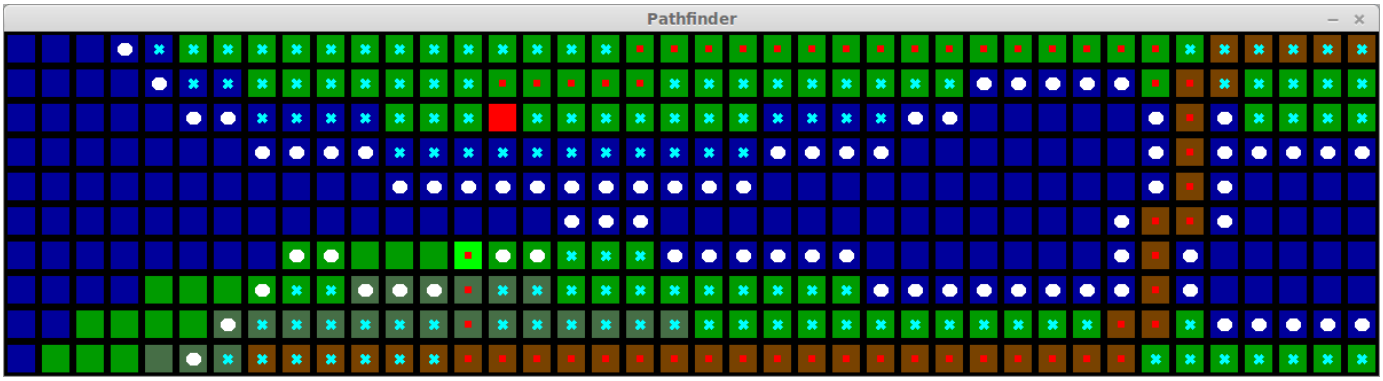

Figure 31: Shortest path using dijkstra with closed and open nodes marked for board 2.4.

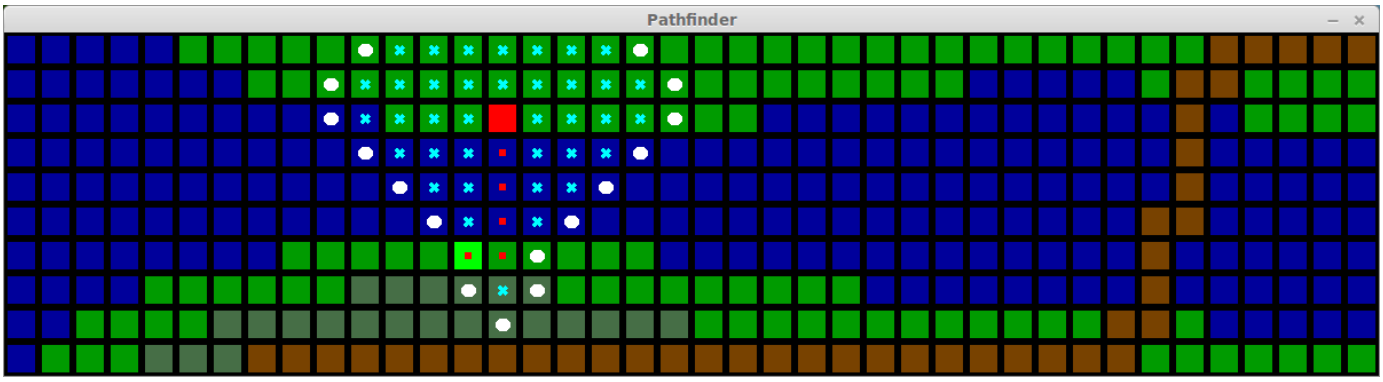

Figure 32: Shortest path using bfs with closed and open nodes marked for board 2.4.