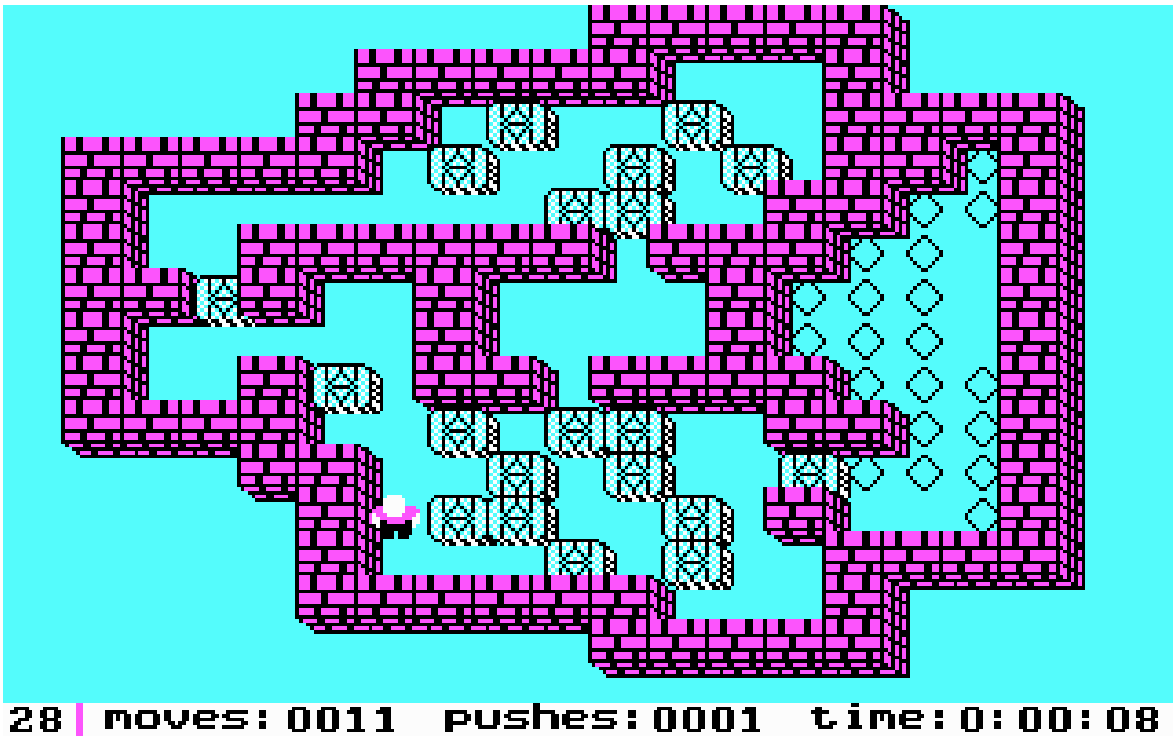
Sokoban solver report

CAB320 Artificial Intelligence semester 1 2020

By James Norman



Background

Sokoban is a computer game invented in 1981 by Japanese game creator Hiroyuki Imabayashi. Sokoban is a puzzle game where the map is a board of squares and you as a player are tasked with moving boxes around in a maze to get them into the target position. The boxes in Sokoban cannot be pulled and cannot be pushed into squares with walls or other boxes. Whilst Sokoban might seem like any other game its reach extends further then just the gaming community with Sokoban having a big impact in the realms of computer science and ai. This is due to solving Sokoban puzzles being shown as being similar to the automated planning required by some autonomous robots. Additionally, Sokoban is interesting to ai as it can be a fairly difficult game, this is due to the huge depth of solutions for such a game and can also be attributed to the fact that any move may leave the puzzle in an unsolvable state making it rather complex for ai.

Aim of assessment

The aim of this assessment is to design and create a program to solve multiple Sokoban puzzles using the guidance of a few helper files and some pseudo code.

Solver overview

To find a solution to the given problem a few informed and uniformed search methods were obtainable. The decision was made that the a star search method would be the most ideal for this assessment and was thus implemented into the code.

The a star search method is an informed search algorithm. This method searches through all of the available paths and picks the one which has the lowest cost, it also considers the path which appears to be the fastest in terms of solve time. This is determined by this equation

$$f(n) = g(n) + h(n)$$

n = last node on the path
g(n) = cost of the path from the start node
h(n) = heuristic that approximates the cheapest path from n to the goal

Using this search allows the solver to search the problem more efficient then various other algorithms as long as the heuristic is done well, the heuristic will be discussed in depth in the next part of the report.

Heuristic

The Heuristic function estimates the distance between the current state and the goal state, each heuristic is designed for a specific search problem which could be Euclidian or Manhattan. Heuristics must be admissible to work and this is only true if they satisfy a set of constraints this being

$$0 \leq h(n) \leq h^*(n) for\ all\ n$$

h(n) = estimated heuristic cost
h^*(n) = true cost to the nearest goal

Heuristics are a great option for a problem like this as they are much more suited to larger problems that humans are unable to solve. The heuristic in the code found the shortest sum of the Manhattan distances between the boxes and the target cells as the code shows in figure 1.

```
def h(self, node):

    warehouse = node.state
    worker = warehouse.worker
    boxes = warehouse.boxes
    targets = warehouse.targets

    heuristic = 0

    for box in boxes:
        boxMin = get_Distance(box, targets)
        heuristic += boxMin

    min = get_Distance(worker, boxes)
    heuristic += min



    return heuristic
```

Figure 1. Heuristic function in python

Performance

To decide which search function to use a few tests were carried out, these tests involved running the code with the a star search method and also running the code with the breadth first search method, after a couple of tests it was found that on average the astar search method was the quickest. It however mainly worked better on bigger warehouses whilst in smaller problems the breadth first got the edge, but as the time difference for smaller problems were so slim it gave the decisive edge to the a star method the table below summarises the two methods performance.

Search Type	No of Boxes	No. of Targets	Warehouse Size	Average Time Taken to Solve over 3 Runs
Breadth First Graph Search	2	2	63	0.04599 seconds
A* Search	2	2	63	0.04838 seconds
Breadth First Graph Search	4	4	79	220.74 seconds
A* Search	4	4	79	160.17 seconds

Table 1. Performance comparison between A* search and breadth first search methods. (note all tests passed)

Conclusion

To conclude in this assessment a code was designed using the a star search method to find a way to solve Sokoban puzzles in a completely autonomous way and in the quickest time possible, whilst there might have been ways to further optimise the code using threading or various other python functions the code ran in an acceptable time and did its desired purpose using the a star search method.