The states:

In my model states consists of numbers and arrays which is stored inside an array as:

[FirstBoxPosition, SecondBoxPosition, Mode, Configuration]

Is an example of one state.

Mode:

If the block is perpendicular to surface it has a Mode of 0, if its parallel to the surface it has a mode of 1. Examples:

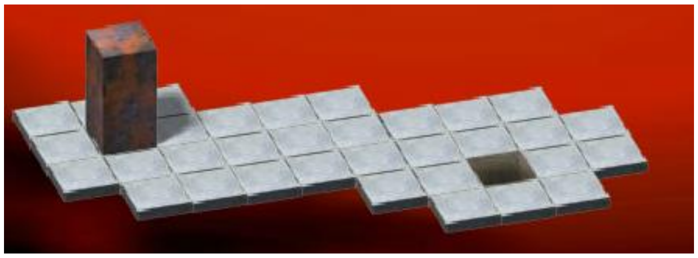


Figure 1: Mode 0

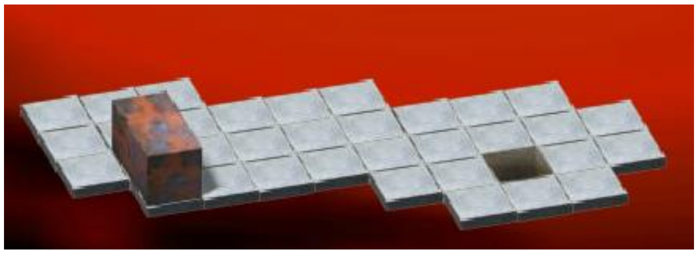


Figure 2: Mode 1

Configuration:

If the block is parallel to x-axis it has a Configuration of 0, if its parallel to the y-axis it has a Configuration of 1.

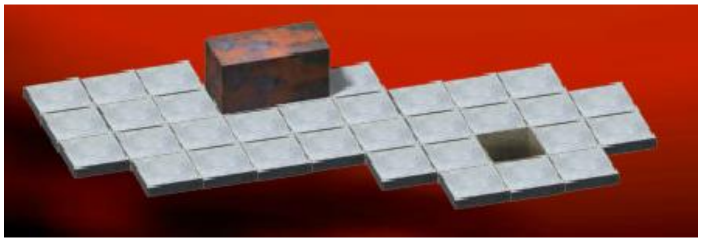


Figure 3: Configuration 0

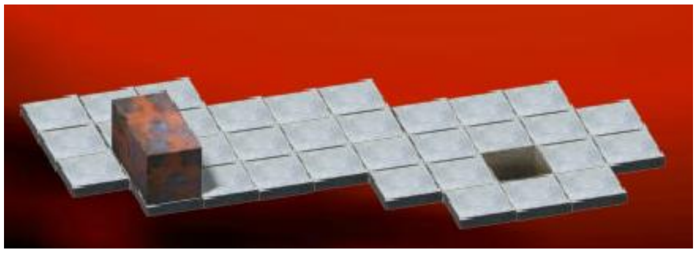


Figure 4: Configuration 1

FirstBoxPosition:

Considering matrix indices (going towards x-axis increases, going towards down of the y-axis increases), FirstBoxPosition is the coordinate array of one of the two boxes that the block has the small coordinates. For instance, in Figure 3, FirstBoxPosition is the coordinate array of the box which is at more of left hand side of the two boxes.

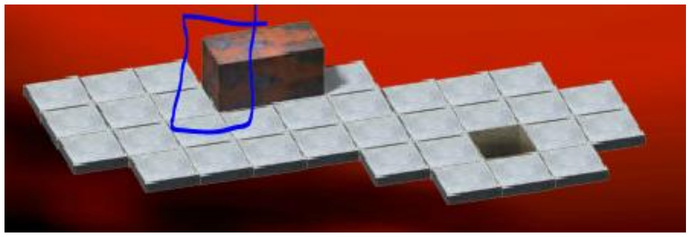


Figure 5. Blue Boardered Box is FirstBoxPosition's Coordinate Array

SecondBoxPosition:

Considering matrix indices (going towards x-axis increases, going towards down of the y-axis increases), SecondBoxPosition is the coordinate array of one of the two boxes that the block has the larger coordinates. For instance, in Figure 3, SecondBoxPosition is the coordinate array of the box which is at more of right-hand side of the two boxes. If the block is in mode 0 such as if Figure 1, SecondBoxPosition is taken as an empty array [ ].

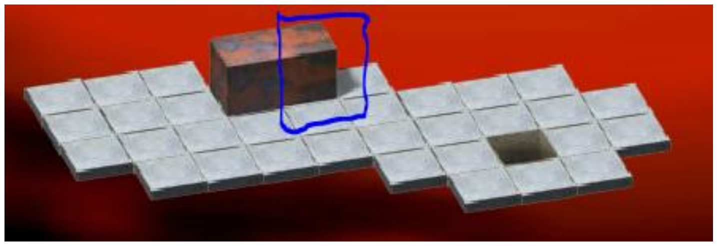


Figure 5. Blue Boardered Box is SecondBoxPosition's Coordinate Array

Successor State Function:

Successor State Function looks at the possible motion of the block which is [MoveUp, MoveRight, MoveDown, MoveLeft]. It checks if the box could perform the action by looking at the board and checking if there is a ground on the coordinates it is suppose to move. If so, it checkes if the previous node has been visited with the same state before which is to stop infinite loops that might occur if we don't keep a visited states list, if not, it calculates the cost related to the state if it the action is performed to that state and puts the cost and the state into the priority queue.

Initial State:

Inıtial state is the block's starting position (coordinates of two blocks), mode, config as:

[FirstBoxPosition, SecondBoxPosition, Mode, Configuration]

Goal Test:

Check if the state is in the following state:

[FirstBoxPosition == Goal Coordinates, SecondBoxPosition == [ ] , Mode == 0, Config == 1 or 0 (Doesn't Matter)]

Step Cost:

UCS:

For the step cost there are two functions. For UCS Search The cost function is the following:

g(n) = g(n – 1) + 1 where g(0) is 1 (*)

The reason for this is that moving the box around should only take 1 cost in each step which would add up when consecutive motion is performed. The goal is to reach the goal position with least effort.

A* Search:

For the A* Search we need a Heuristic function:

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

where f(n) is the Heuristic Function, g(n) is the is the cost of the path to node n and h(n) is the heuristic estimate of the cost of getting to a goal node from n.

g(n) is again *,

For h(n) I have choicen the mathematical distance of the two points between the center of the block and the goal position which lead to first finding the center of the block is the block is in mode 1.

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figure 6: Distance Formula

If the block is in mode 0 its center is the coordinate of the FirstBoxPosition otherwise it's the average of each coordinate of two blocks.

$$\text{Midpoint} = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}\right)$$

Figure 7: Center Of Two Points

Proving Heuristic Function is Admissible:

Let h*(n) be the cost of an optimal path from n to a goal node (∞ if there is no path). Then an admissible heuristic satisfies the condition: h(n) ≤ h*(n).

Lets think of our function in Figure 6.

In the best case of our block position (when the block could move straight in axis' without having to change axis to get over some areas of the ground so that it spends the optimal cost to get to the goal point) we can travel in straight within two axis in order and win such as ["Right", "Right", "Right", "Down", "Down"].

Our path will look like:

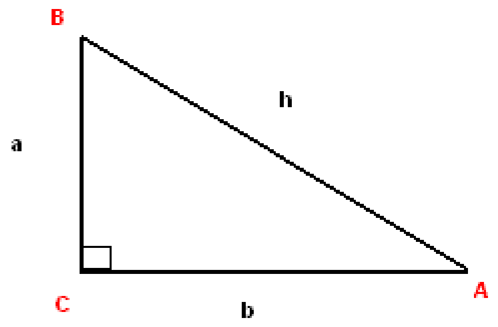


Figure 8: Perpendicular Triangle

And from the triangle inequality:

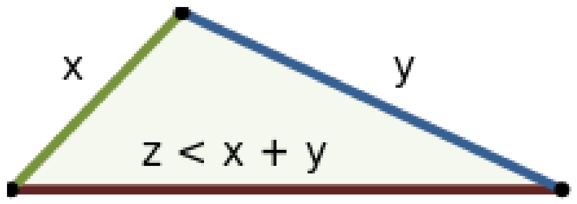


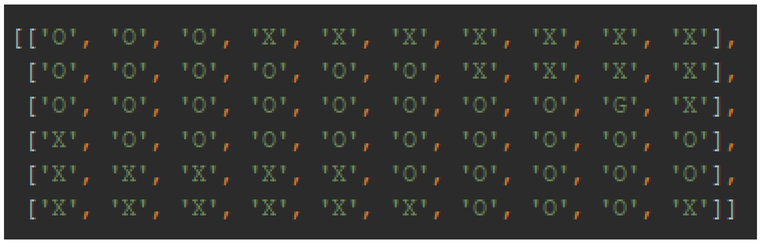
Figure 9: Triangle Inequality

By using Figure 8 and 9 we can see that our h(n) which will be h, which is always smaller than the actual cost which is a + b which means that condition h(n) ≤ h*(n) always hold.

Performance Comparison:

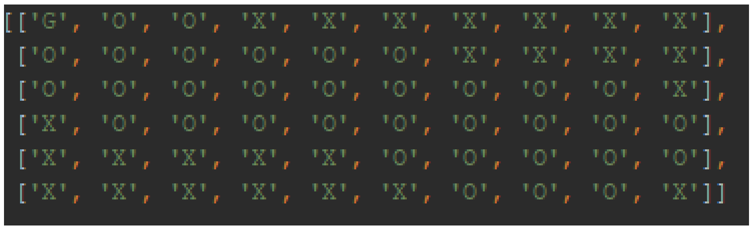
Lets compare the search algorithms for 3 distinct boards with goals with respect to Time and Peak Memory Consumption:

1:



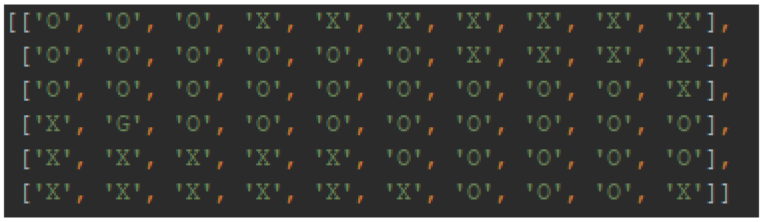
```
[['O', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
 ['O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
 ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'G', 'X'],
 ['X', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],
 ['X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'O', 'O'],
 ['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']]
```

2:



```
[['G', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
 ['O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
 ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'X'],
 ['X', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],
 ['X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'O', 'O'],
 ['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']]
```

3:



```
[['O', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
 ['O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
 ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'X'],
 ['X', 'G', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O'],
 ['X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'O', 'O'],
 ['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']]
```

*A*Search*	*Time*	*Memory*
1	0.1166	71259 KiB
2	0.0249	45115 KiB
3	0.1027	65587 KiB

UCS	*Time*	*Memory*
1	0.1675	85779 KiB

2	0.0518	52907 KiB
3	0. 1416	76019 KiB

As we can see A* Search is both faster and less memory using algorithm than UCS, this is expected because since A* Search is using a heuristic function its able to make more logical decisions than UCS algorithm which leads to faster approach to the goal leading to decreased execution time and making the priority queue less sized.