Vincentius Aditya Sundjaja
COMP3702
45610099

COMP3702/7702 ARTIFICIAL INTELLIGENCE

ASSIGNMENT 1 REPORT

Sokoban – Discrete Search

I. Agent Design Problem

1. Action Space (A)
   Move the player to left (L), right (R), up (UP), down (DOWN) if the next cell is not an obstacle or 2 consecutive boxes or a deadlock state

2. Percept Space (P)
   A 2-D character array with the row and column representing the (x, y) position on the map. A wall is marked with '#', a free space is marked with ' ', a box is marked with 'B', a target is marked with 'T', a player is marked with 'P', a box on top of a target is marked with 'b', and a player on top of a target is marked with a 'p'. All the (x, y) positions of boxes and targets are stored in separate lists of tuples and the player position is stored as a tuple.

3. State Space (S)
   All possible combinations of player, box, and free space on the map.

4. World dynamics (T: S x A → S')
   Given a possible action, the percept space is changed accordingly and the list storing the (x, y) positions of boxes and targets is adjusted as well as the tuple for the player position

5. Perception Function (Z: S → P)
   For this problem, the percept is the same as the current state

6. Utility Function
   For this problem, I don't see how we can use utility function. Except if heuristic is considered as a utility function

II. Agent Type

1. Discrete → all state / action / percept spaces are discrete, meaning they have fixed number of combinations.
2. Deterministic → it is deterministic because the next state can be known given the knowledge of the previous state and the action taken
3. Fully Observable → Yes because, the agent knows and has access to all information in the environment to solve the problem
4. Static → the world will remain unchanged unless the player/agent moves
5.

Vincentius Aditya Sundjaja
COMP3702
45610099

III. A* search Heuristic

The heuristic I used for A* search is combination of Goal Pull Distance [1] and Manhattan Distance.

- Goal Pull Distance is used to calculate the distances from all possible cells (including box cells) to a goal/target. So, for each target, we "pull" a box from a target by checking all its 4 direction (up, down, right, left) and see whether a box put in that position can be pushed onto the goal. This algorithm makes use of the breadth first search algorithm. The result of this is used for calculating heuristic and marking cells of a simple deadlock.

#	#	#	#	#	#
#	**T**	1	2	inf	#
#	1	2	3	inf	#
#	2	3	4	inf	#
#	inf	inf	inf	inf	#
#	#	#	#	#	#

Here is an example of a goal pull distance, each cell is marked with the distance it takes to get to the target and the one marked 'inf' means that it's unreachable (deadlock).

- Then to calculate the heuristic, we calculate the minimum distance of a box to a target by checking it using the goal pull distance result. We sum up the result and then we also calculate the minimum distance of the player to a box. Calculating this is done by using the Manhattan Distance. We sum up the result again to get the final heuristic cost.
- This heuristic is admissible because by the sum of minimum distance of getting all boxes to a target will never overestimate the sum of real distance of getting all the boxes to a target. As for the distance from player to the nearest box will also not overestimate because Manhattan Distance is used to calculate it.

IV. Performance Result

My solution is not able to solve all the given testcases (4box_m2 got an exception of MemoryError, 4box_m3 exceeded the time limit even though solution is found).

1. 1box_m1

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\1box_m1.txt
UCS DONE
Generated:  234
In Fringe:  27
Visited:  80
Time:  0.0054726
7 steps: d,l,l,u,l,d,d

A* DONE
Generated:  58
In Fringe:  27
Visited:  18
Time:  0.002507699999999999
7 steps: l,d,d,r,d,l,l
```

2. 1box_m2

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\1box_m2.txt
UCS DONE
Generated:  475
In Fringe:  36
Visited:  179
Time:  0.0102532
26 steps: l,l,u,l,l,d,u,r,r,d,d,l,l,l,d,l,l,u,u,r,d,d,l,d,r,r

A* DONE
Generated:  304
In Fringe:  35
Visited:  114
Time:  0.009893
26 steps: l,l,u,l,l,d,u,r,r,d,d,l,l,l,d,l,l,u,u,r,d,d,l,d,r,r
```

3. 1box_m3

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\1box_m3.txt
UCS DONE
Generated:  969
In Fringe:  17
Visited:  399
Time:  0.0198978
57 steps: u,r,u,u,u,u,u,r,r,r,r,d,d,l,d,d,d,d,r,r,u,l,d,l,u,u,u,l,u,r,d,r,u,u,u,r,u,l,u,l,d,r,d,l,l,l,u,l,d,d,d,d,d,l,d,r

A* DONE
Generated:  470
In Fringe:  37
Visited:  193
Time:  0.0163416
57 steps: u,r,u,u,u,u,u,r,r,r,r,d,d,d,l,d,d,d,r,r,u,l,d,l,u,u,u,l,u,r,d,r,u,u,u,l,u,u,r,d,r,d,l,l,l,l,u,l,d,d,d,d,d,l,d,r
```

4. 2box_m1

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\2box_m1.txt
UCS DONE
Generated:  91
In Fringe:  0
Visited:  43
Time:  0.0043663
13 steps: u,u,l,l,d,d,l,l,u,l,l,d,r

A* DONE
Generated:  87
In Fringe:  4
Visited:  41
Time:  0.004861
13 steps: u,u,l,l,d,d,l,l,u,l,l,d,r
```

5. 2box_m2

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\2box_m2.txt
UCS DONE
Generated:  19284
In Fringe:  89
Visited:  7614
Time:  0.4845818
115 steps: u,u,r,u,u,u,u,r,r,r,r,d,d,l,d,d,d,d,d,r,r,u,l,d,l,u,u,u,u,l,u,r,d,r,u,u,u,l,u,u,r,d,r,d,l,d,d,d,l,d,d,d,r,r,u,l,d,l,u,u,u,l,u,r,d,r,u,u,u,l,l,l,u,l,d,d,d,d,d,d,u,u,u,u,u,
u,r,r,r,u,u,r,d,r,d,l,l,l,l,u,l,d,d,d,d,d,d,l,d,r,l,d,r

A* DONE
Generated:  16038
In Fringe:  349
Visited:  6372
Time:  0.7752700000000001
115 steps: u,r,u,u,u,u,u,u,r,r,r,r,d,d,d,l,d,d,d,d,d,r,r,u,l,d,l,u,u,u,u,l,u,r,d,r,u,u,u,l,u,u,r,d,r,d,l,d,d,d,l,d,d,d,r,r,u,l,d,l,u,u,u,l,u,r,d,r,u,u,u,l,l,l,u,l,d,d,d,d,d,d,l,d,r,u,u,u,
u,u,u,r,r,r,u,u,r,d,r,d,l,l,l,l,u,l,d,d,d,d,d,d,l,d,r
```

6. 2box_m3

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\2box_m3.txt
UCS DONE
Generated:  56586
In Fringe:  156
Visited:  22971
Time:  1.4822157
115 steps: u,u,r,u,u,u,u,u,r,r,r,d,r,r,r,u,u,r,r,r,d,l,l,u,l,d,r,d,l,l,l,r,r,d,d,d,d,d,l,l,u,l,d,l,u,u,u,r,u,u,u,r,u,l,l,l,l,u,l,d,d,d,d,d,d,d,u,u,u,u,u,u,u,r,r,r,r,d,d,d,d,l,l,
u,r,d,r,u,u,u,r,u,l,l,l,l,u,l,d,d,d,d,d,d,d,l,d,r,l,d,r

A* DONE
Generated:  44561
In Fringe:  911
Visited:  18107
Time:  2.4025393
115 steps: u,u,r,u,u,u,u,u,u,r,r,r,d,r,r,r,u,r,u,r,r,d,l,l,u,l,d,r,d,l,l,l,r,r,d,d,d,d,d,d,l,l,u,l,d,l,u,u,u,r,u,u,u,r,u,l,l,l,l,u,l,d,d,d,d,d,d,l,d,r,u,u,u,u,u,u,u,r,r,r,r,d,d,d,d,
l,l,u,r,d,r,u,u,u,r,u,l,l,l,l,u,l,d,d,d,d,d,d,d,l,d,r
```

7. 3box_m1

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\3box_m1.txt
UCS DONE
Generated:  4042
In Fringe:  81
Visited:  1536
Time:  0.1191611
27 steps: u,u,r,r,u,r,d,d,r,d,l,u,u,l,l,l,l,l,d,d,l,l,u,l,l,d,r

A* DONE
Generated:  2986
In Fringe:  175
Visited:  1137
Time:  0.1803827
27 steps: u,u,r,r,u,r,d,d,r,d,l,u,u,l,l,l,l,l,d,d,l,l,u,l,l,d,r
```

8. 3box_m2

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\3box_m2.txt
UCS DONE
Generated:  403095
In Fringe:  2773
Visited:  152212
Time:  13.9112222
66 steps: l,d,l,l,u,l,u,l,l,d,r,u,r,d,d,d,l,d,r,r,u,u,r,u,u,u,r,r,r,r,u,r,d,d,d,u,u,l,l,l,u,l,u,l,l,d,r,u,r,d,l,d,r,r,r,r,u,r,d,d,r,d,l,r,d,l

A* DONE
Generated:  253917
In Fringe:  10928
Visited:  95129
Time:  20.760102200000002
66 steps: l,d,l,l,u,l,u,l,l,d,r,u,r,d,d,d,l,d,r,r,u,u,r,u,u,u,r,r,r,r,u,r,d,d,r,d,l,u,u,l,l,u,l,l,u,l,l,d,r,u,r,d,l,d,r,r,r,r,u,r,d,d,r,d,l,r,d,l
```

9. 4box_m1

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\4box_m1.txt
UCS DONE
Generated:  61653
In Fringe:  44
Visited:  24933
Time:  2.4349825999999997
89 steps: u,u,l,l,l,d,d,l,u,u,u,u,u,l,u,r,r,r,l,l,d,d,d,d,d,l,l,u,r,d,r,u,u,u,u,u,l,u,r,r,l,d,d,d,d,r,r,r,u,u,l,d,r,d,l,l,d,l,l,d,r,u,r,d,d,d,d,r,r,r,u,u,l,d,r,d,l,l,d,l,u,u,u,l,u,r,d,d,d,r,r,r,u,u,l,d,r,d,l,l,d,l,u,u,u,u,l,u,r,d,d,d,d,r,r,r,u,u,l,d,r,d,l,l,d,l,u,u,u
,u

A* DONE
Generated:  60199
In Fringe:  379
Visited:  24331
Time:  6.1334697
89 steps: u,u,l,l,l,d,d,l,u,u,u,u,u,l,u,r,r,r,l,l,d,d,d,d,r,r,r,u,u,l,d,r,d,l,l,l,d,l,u,u,u,u,l,u,r,r,l,d,d,d,d,d,d,l,l,u,r,d,r,u,u,u,u,u,l,u,r,d,d,d,d,r,r,r,u,u,l,d,r,d,l,l,d,l,u,u,u
,u
```

Vincentius Aditya Sundjaja
COMP3702
45610099

10. 4box_m2

```
During handling of the above exception, another exception occurred:

MemoryError
```

11. 4box_m3

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\4box_m3.txt
UCS DONE
Generated:  3014041
In Fringe:  108
Visited:  1102930
Time:  127.1301673
84 steps: u,l,l,l,l,l,u,r,u,l,l,l,l,d,l,u,u,l,u,r,r,l,d,d,d,r,d,d,r,u,d,d,r,r,u,u,l,l,r,r,u,r,u,l,l,l,l,d,l,u,u,l,u,r,d,d,d,d,d,r,u,r,u,l,d,l,u,u,u,d,r,r,d,d,r,r,u,u,l,l,l,d,l,u
```

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python solver.py testcases\4box_m3.txt
A* DONE
Generated:  2798488
In Fringe:  28838
Visited:  1018903
Time:  356.1076897
84 steps: u,l,l,l,l,l,u,r,u,l,l,l,l,d,l,u,u,l,u,r,r,l,d,d,d,d,r,d,r,u,d,d,r,r,u,u,l,l,l,d,l,u,u,u,u,l,u,r,d,d,r,r,r,r,d,r,d,l,l,l,u,d,l,d,l,u,u,u,u,d,r,d,d,r,r,r,u,u,l,l,l,d,l,u
```

12. 6box_m2

```
V:\Vincent\KULIAH\AUSSIE\2019_SEM2\COMP3702\prac\comp3702_python\assignment1>python ucs2.py testcases\6box_m2.txt
UCS DONE
Generated:  611964
In Fringe:  15028
Visited:  219565
Time:  31.1978951
35 steps: u,r,r,u,u,r,r,r,l,l,d,r,r,d,d,l,l,u,r,r,l,l,l,l,l,u,u,r,r,l,d,l,d,r,r

A* DONE
Generated:  319090
In Fringe:  27881
Visited:  113800
Time:  53.6995378
37 steps: u,u,r,r,d,r,r,r,l,l,l,u,u,r,r,r,l,l,d,r,r,l,l,l,d,d,l,u,l,u,r,r,l,l,u,r,r
```

Performance Comparison Table

	Nodes Generated		Nodes in Fringe		Nodes Visited		Algorithm Run Time (seconds)		Number of Steps	
	UCS	A*	UCS	A*	UCS	A*	UCS	A*	UCS	A*
1box_m1	234	58	27	27	80	18	0.005	0.002	7	7
1box_m2	475	304	36	35	179	114	0.0102	0.0098	26	26
1box_m3	969	470	17	37	399	193	0.019	0.016	57	57
2box_m1	91	87	0	4	43	41	0.0043	0.0048	13	13
2box_m2	19284	16038	89	349	7614	6372	0.484	0.775	115	115
2box_m3	56586	44561	156	911	22971	18107	1.482	2.402	115	115
3box_m1	4042	2986	81	175	1536	1137	1.1191	0.1803	27	27
3box_m2	403095	253917	2773	10928	152212	95129	13.911	29.760	66	66
4box_m1	61653	60199	44	397	24933	24311	2.434	6.133	89	89
4box_m2	MemoryError									
4box_m3	3014041	2798488	108	28838	1102930	1018903	127.130	356.107	84	84
6box_m2	611964	464723	15028	32191	219565	166275	33.52	88.39	35	35

Evaluation

- For all test cases, using the A* search results in less generated and less visited nodes. For all test cases, except 4box_m2, they are all able to find the optimal solutions. However, for test cases starting from 2box_m1, the run time of the A* search is longer than UCS. I am assuming because the function for calculating the heuristic in A* is still time consuming and not efficient. However, they both still gives the optimal solution.

- For 4box_m2, it couldn't be solved with my program. It reaches an exception which is MemoryError. This happens because too many nodes are generated (a lot of them might be deadlocks and unnecessary moves).

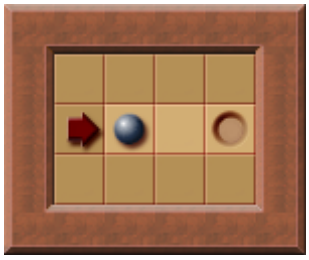
- For 4box_m3, both UCS and A* exceeded the time limit given (number_of_boxes * 30s). However, both can find the optimal solution. Possible reason is that the get successor method is still not optimal, deadlock squares not detected, etc.

- For 6box_m2, both UCS and A* are able to find the optimal solution within the time limit. Even though the boxes are more than the test case in 4box_m2 and m3, solution is still able to be find. Possible reason is that because the map is not as complex as 4box_m2 and m3. The more complex a map, the possibility of a deadlock occurring increases. As my program only detect simple deadlocks, complex Sokoban map will need very long time or even impossible to solve.

- Faster run time can be achieved by detecting more deadlock states, by optimizing the functions used such as get successor, and by fixing the heuristic so it can work for large number of boxes as well.

V. Deadlock Detection

The deadlock detection that I used in my solution is only able to detect simple deadlocks. From [2], simple deadlocks or dead square deadlocks are squares in a level where when a box is pushed into them, the box cannot be pushed into a goal anymore.



Source: http://sokobano.de/wiki/images/SimpleDeadlockExample.png

However, there are still more types of deadlock that my solution hasn't checked (freeze, coral, etc.), which makes my solution for large number of boxes slow.

Vincentius Aditya Sundjaja
COMP3702
45610099

So, to get the simple deadlock squares, I used the Goal Pull Distance again. From the table below, the ‘inf’ means that a box in that position cannot be pushed to a goal. From Figure 2, there are several squares that are marked possible to go to and impossible to go to. This will still be marked possible to go to. So, by gathering all the squares that have ‘inf’ value, I will have a list of deadlock squares and whenever a successor is generated, it first checks if any box is in one the deadlock squares.

#	#	#	#	#	#
#	**T (0)**	1	2	inf	#
#	1	2	3	inf	#
#	2	3	4	inf	#
#	inf	inf	inf	inf	#
#	#	#	#	#	#

#	#	#	#	#	#
#	**T (0)**	1 / inf	2 / inf	inf	#
#	1 / 2	2 / 3	3 / 4	inf	#
#	2 / 1	3 / 2	4 / 3	inf	#
#	T (0)	inf / 1	inf / 2	inf	#
#	#	#	#	#	#

Possible solution to find other deadlocks

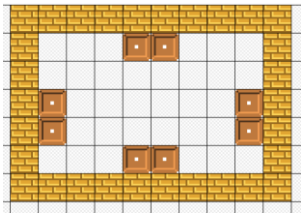
```
for box in self.boxes:
    box_0 = box[0]
    box_1 = box[1]

    top = (box_0 + surrounding_coordinates["top"][0], box_1 + surrounding_coordinates["top"][1])
    top_r = (box_0 + surrounding_coordinates["top_r"][0], box_1 + surrounding_coordinates["top_r"][1])
    right = (box_0 + surrounding_coordinates["right"][0], box_1 + surrounding_coordinates["right"][1])
    bot_r = (box_0 + surrounding_coordinates["bot_r"][0], box_1 + surrounding_coordinates["bot_r"][1])
    bot = (box_0 + surrounding_coordinates["bot"][0], box_1 + surrounding_coordinates["bot"][1])
    bot_l = (box_0 + surrounding_coordinates["bot_l"][0], box_1 + surrounding_coordinates["bot_l"][1])
    left = (box_0 + surrounding_coordinates["left"][0], box_1 + surrounding_coordinates["left"][1])
    top_l = (box_0 + surrounding_coordinates["top_l"][0], box_1 + surrounding_coordinates["top_l"][1])

    # CHECK IF ANY BOX IS NEXT TO ANOTHER BOX ADJACENT TO A WALL
    if (self.state[left[0]][left[1]] == "#" and self.state[top_l[0]][top_l[1]] == "#"   # left side
            and (self.state[top[0]][top[1]] in unmovable_obj)) or \
            (self.state[left[0]][left[1]] == "#" and self.state[bot_l[0]][bot_l[1]] == "#"
             and (self.state[bot[0]][bot[1]] in unmovable_obj)) or \
            (self.state[right[0]][right[1]] == "#" and self.state[top_r[0]][top_r[1]] == "#"    # right side
             and (self.state[top[0]][top[1]] in unmovable_obj)) or \
            (self.state[right[0]][right[1]] == "#" and self.state[bot_r[0]][bot_r[1]] == "#"
             and (self.state[bot[0]][bot[1]] in unmovable_obj)) or \
            (self.state[top[0]][top[1]] == "#" and self.state[top_r[0]][top_r[1]] == "#"    # top side
             and (self.state[right[0]][right[1]] in unmovable_obj)) or \
            (self.state[top[0]][top[1]] == "#" and self.state[top_l[0]][top_l[1]] == "#"
             and (self.state[left[0]][left[1]] in unmovable_obj)) or \
            (self.state[bot[0]][bot[1]] == "#" and self.state[bot_r[0]][bot_r[1]] == "#"    # bot side
             and (self.state[right[0]][right[1]] in unmovable_obj)) or \
            (self.state[bot[0]][bot[1]] == "#" and self.state[bot_l[0]][bot_l[1]] == "#"
             and (self.state[left[0]][left[1]] in unmovable_obj)):

        return True
```

The above function checks if two boxes are located consecutively and adjacent to a wall such as:



VI. References

[1] https://baldur.iti.kit.edu/theses/SokobanPortfolio.pdf
[2] http://sokobano.de/wiki/index.php?title=Deadlocks#Dead_square_deadlocks
[3] http://sokobano.de/wiki/index.php?title=How_to_detect_deadlocks