COMP3702 Assignment 1 Report – Angus Trusler

Agent Definition:

State Space: All permutations of valid positions of the Boxes, and the player

Action Space: Move player position up/down/left/right

Transition Function: T(S, A) = S' – the state transitions from one state, to another when the player makes a move.

Percept Space / Percept function: (not needed here: world fully observable)

Agent Description:

Discrete: There exists a finite number of combinations in which the game's objects may be positioned for any given board.

Fully observable: At any given moment in time, the agent has access to all the information about the board.

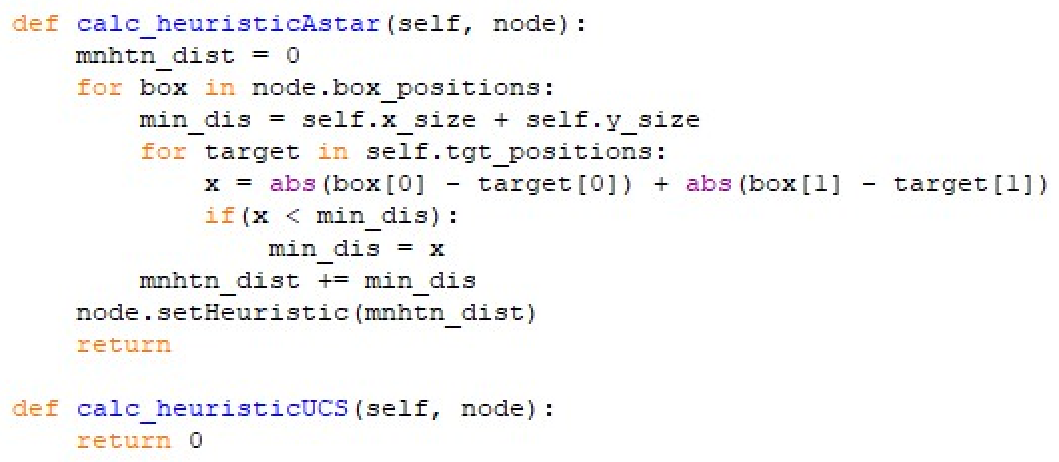
Deterministic: The nature of each move is at the discretion of the agent, and the agent's moves are the only factor in determining the future board layout.

Static: Nothing about the board changes until the agent makes a move.

Each board state is a node on the tree, and each child is a succussing state after the agent makes a move.

Heuristic Explanation and Justification

In my solution the heuristic is as follows:



```
def calc_heuristicAstar(self, node):
    mnhtn_dist = 0
    for box in node.box_positions:
        min_dis = self.x_size + self.y_size
        for target in self.tgt_positions:
            x = abs(box[0] - target[0]) + abs(box[1] - target[1])
            if(x < min_dis):
                min_dis = x
        mnhtn_dist += min_dis
    node.setHeuristic(mnhtn_dist)
    return

def calc_heuristicUCS(self, node):
    return 0
```

The heuristic is the sum of the Manhattan distances from each box to their nearest target. It ignores obstacles, and boxes may share a target. When put into the priority queue, the queue class adds this heuristic value and the depth of the node to determine the node's position in the queue (Worst case O(n) where n is queue length). This is why the UCS function has a heuristic function that simple returns 0: it allows the same node class to be used for different algorithms.

It isn't perfect, but this heuristic is admissible, as it never overestimates the cost of moving a box to a target (it severely underestimates) and so still allows the agent to find the optimal path. A few potential different strategies were considered:

- The lists for target and box positions could be maintained in a sorted state, so the heuristic still utilises a Manhattan distance to a nearest target not already calculated to a closer box. This would maintain the admissibility feature of the current function but provide a more accurate and dynamic output for better prioritisation of nodes.
- The program could precalculate a path for each box to its nearest available target and sum the total distances. This would maintain admissibility but would require added tracking for detecting when the box is moving along its path and come with its own computational concerns.
- The heuristic could take into account the player's relative position to a box not currently on a target: again, this would maintain admissibility, and add responsiveness to when the agent has achieved one box goal and needs to move onto another.

Ultimately, the heuristic I have created remains superior to no heuristic in that it dramatically cuts down search time and space, as covered in the next section.

Performance comparison of Uniform Cost and A* search

The following comparisons are made on the generated outputs of the algorithms' reaction to the provided map "2box_m2.txt", with an optimal path length of 115.

1. **Number of Nodes Generated**

   A*: 32022 UCS: 35334 Difference: 3312 nodes

   Even with a heuristic that's suboptimal in terms of accuracy, it's clear that it guides the agent well enough to finding the optimal solution while exploring 3312 fewer nodes. This is a difference of approximately 10%, which when on search spaces of a larger scale, could mean seconds of time difference.

2. **Number of nodes on fringe when search terminates**

   A*: 484 UCS: 179 Difference: 305 nodes

   A larger fringe means the agent is prioritising nodes differently. This is because A* uses a heuristic to ignore already generated nodes that according to the heuristic are less likely to result in an optimal solution.

   A theoretically perfect algorithm would have a fringe of (path_length * 4): as a perfect agent would pick the correct move out of 4 possible moves, every time for the length of the path from the initial state to the goal state.

3. **Number of explored nodes when search terminates**
   A*: 31538 UCS: 35155 Difference: 3617
   Again, owing to the use of a heuristic, the A* algorithm locates the optimal path earlier than the UCS algorithm.

4. **Algorithm run time**
   A*: 8.99 seconds UCS: 9.43 seconds Difference: 0.44 seconds
   While this doesn't seem like much, these results are repeatable across all maps. At larger scales, these differences become more pronounced.

Deadlock function

My solution does not implement a deadlock function. One way I would implement one in future, is to add a ‘deadlocked’ flag on any node as it is explored. When exploring nodes, the agent checks if each box is in a deadlocked position (walls on at least two perpendicular sides and not on a target).

```
   #
  B#
####
```

A deadlocked position would also exist wherein boxes are lined up together, and their removal is impossible. Eg:

```
  BB
########
```

If found, the node would be flagged, and heuristic value would be set to integer_max – depth (so it is permanently at the end of the queue). The agent would not generate children for it and skip to the next node in the queue. If the next node is a previously marked deadlocked node, the program would terminate as a solution is not possible.

Of course, any map with a deadlocked starting position could be detected ahead of time, and the program refuse to start searching.

Areas identified for future improvement

A deadlock function would be the first area I would improve. Maps have many deadlocking areas, and a way to programmatically exclude them from the search space would yield massive improvement.

Another area of improvement would be preventing the agent from reacting strangely to box collisions. It appears to struggle greatly when the agent aligns two boxes next to one another and tries to push them.