Derek Pruski (dmp3872) and Kellen Bell (kdb9520)

CSCI 331 Project 2 Writeup

State Implementation:

Init Function:

The init function takes in the initial state of the board as initial and number of attendants as cars_per_action. These variables are stored in the problem class for future functions to reference.

Actions Function:

The action function takes in the current state of the board and generates a list of all possible actions as a set of car/action pairs. We start by looking at each individual car and storing all possible moves and new coordinate positions. Ex. [0, 'down', (0,1)]. Then using the python itertools.combinations function we get all possible combinations passing in the list of possible moves (combinations) and the number of cars allowed to move at one time (self.cars_per_action). To ensure all combinations are valid action sets, we loop through each combo and remove any combo that either moves cars to the same coordinate or moves the same car twice in one action set. After checking that each combination is valid, we add (car, move direction) to a list of unique combinations and return them to the caller.

Result:

The result function takes in the current state and current action set to perform. We start by creating a copy of the state and looping through each move in the action set. For each move we adjust the coordinates of the current car. After all moves are accounted for, we create a new state passing in the modified state and a copy of the barriers. The new state is then returned to the caller.

Goal_test, path_cost, value:

All these functions were not modified for our implementation.

Heuristic Definition:

Our heuristic_dist function calculates the distance from each car to its goal state and sums all these values to produce the heuristic_dist for the given state. We use Pythagorean theorem to calculate the distance for each car, before summing all of these distances. Using our heuristic, we were able to get the best_first_graph_search algorithm to find a solution for a run of 6 car, 3 attendants, and 3 barriers in 32.1 seconds. The best run we got from astar_search was 5 cars, 2 attendants, and 1 barrier.

Branching Factor:

5^m

To calculate the branching factor of a state we would take the number of possible moves for each car (5) and raise that to the number of attendants to cover the possible combinations of attendants and cars.

State Space:

((n x n – b) choose n)^m

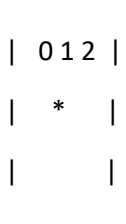
To calculate the state space, we have n x n coordinates on the board, and we need to subtract the number of barriers from the possible open spots. From the remaining spots on the board, we can configure n cars, so we do ((n x n – b) choose n). To consider multiple cars moves at one time, the state space size gets raised exponentially so, we raise this entire equation to the m for attendants. Giving us the final equation of ((n x n – b) choose n)^m

Depth_first_tree_search:

We were unable to produce a result for the depth_first_tree_search because this algorithm follows a single path until there are no new possibilities. However, since the state space is not checked for repeating paths the current car will continue to move back and forth not allowing the algorithm to move on.

Depth_first_graph_search:

-s depth_first_graph_search -c 3 -a 3 -b 1



[{(0, 'stay'), (1, 'stay'), (2, 'down')}, {(0, 'stay'), (1, 'stay'), (2, 'down')}, {(2, 'left'), (0, 'stay'), (1, 'stay')}, {(2, 'left'), (0, 'stay'), (1, 'stay')}, {(1, 'right'), (0, 'stay'), (2, 'up')}, {(2, 'stay'), (1, 'down'), (0, 'stay')}, {(2, 'stay'), (1, 'down'), (0, 'stay')}, {(2, 'stay'), (0, 'stay'), (1, 'left')}, {(2, 'stay'), (0, 'right'), (1, 'left')}, {(0, 'stay'), (1, 'stay'), (2, 'up')}, {(1, 'up'), (0, 'right'), (2, 'stay')}, {(0, 'stay'), (1, 'stay'), (2, 'right')}, {(1, 'up'), (0, 'down'), (2, 'stay')}, {(0, 'stay'), (1, 'stay'), (2, 'right')}, {(2, 'stay'), (0, 'down'), (1, 'right')}, {(0, 'stay'), (1, 'stay'), (2, 'down')}, {(1, 'right'), (0, 'left'), (2, 'stay')}, {(0, 'stay'), (1, 'stay'), (2, 'down')}, {(2, 'stay'), (1, 'down'), (0, 'left')}, {(2, 'left'), (0, 'stay'), (1, 'stay')}, {(2, 'stay'), (1, 'down'), (0, 'up')}, {(2, 'stay'), (1, 'stay'), (0, 'up')}, {(2, 'stay'), (0, 'right'), (1, 'stay')}, {(2, 'stay'), (0, 'right'), (1, 'stay')}, {(2, 'left'), (0, 'down'), (1, 'stay')}, {(0, 'stay'), (1, 'stay'), (2, 'up')}, {(0, 'stay'), (1, 'stay'), (2, 'up')}, {(0, 'stay'), (1, 'stay'), (2, 'right')}, {(0, 'stay'), (1, 'left'), (2, 'right')}, {(2, 'stay'), (0, 'down'), (1, 'stay')}, {(0, 'stay'), (1, 'left'), (2, 'down')}, {(2, 'stay'), (0, 'left'), (1, 'stay')}, {(1, 'up'), (0, 'stay'), (2, 'down')}, {(2, 'stay'), (0, 'left'), (1, 'stay')}, {(1, 'up'), (2, 'left'), (0, 'stay')}, {(1, 'stay'), (2, 'right'), (0, 'up')}, {(0, 'stay'), (1, 'stay'), (2, 'up')}, {(0, 'stay'), (1, 'stay'), (2, 'up')}, {(2, 'left'), (0, 'down'), (1, 'stay')}, {(2, 'stay'), (1, 'down'), (0, 'stay')}, {(2, 'left'), (0, 'right'), (1, 'stay')}, {(2, 'stay'), (1, 'down'), (0, 'stay')}, {(0, 'right'), (1, 'stay'), (2, 'down')}, {(1, 'right'), (0, 'stay'), (2, 'stay')}, {(0, 'stay'), (1, 'stay'), (2, 'down')}]

elapsed time: 0.01594376564025879 seconds

Breadth_first_graph_search:

-s breadth_first_graph_search -c 3 -a 3 -b 1

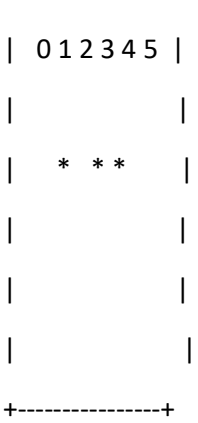
```
+----------+
|  0 1 2  |
|    *     |
|          |
+----------+
```

[{(2, 'down'), (0, 'down'), (1, 'stay')}, {(0, 'up'), (2, 'down'), (1, 'right')}, {(1, 'down'), (0, 'right'), (2, 'left')}, {(1, 'down'), (0, 'right'), (2, 'left')}, {(1, 'left'), (2, 'up'), (0, 'down')}, {(2, 'down'), (0, 'down'), (1, 'stay')}]

elapsed time: 0.02528691291809082 seconds

Best_first_graph_search:

-s best_first_graph_search -c 6 -a 3 -b 3

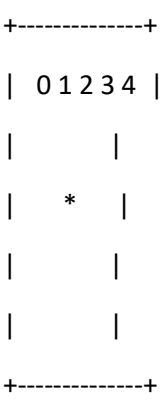


[{(4, 'down'), (2, 'down'), (3, 'down')}, {(2, 'down'), (5, 'left'), (1, 'down')}, {(2, 'down'), (5, 'left'), (0, 'right')}, {(2, 'down'), (0, 'right'), (1, 'right')}, {(2, 'right'), (0, 'stay'), (1, 'down')}, {(2, 'down'), (0, 'down'), (1, 'down')}, {(5, 'left'), (0, 'down'), (1, 'right')}, {(5, 'down'), (0, 'down'), (1, 'down')}, {(5, 'down'), (0, 'right'), (1, 'right')}, {(5, 'down'), (0, 'right'), (1, 'down')}, {(5, 'left'), (0, 'down'), (3, 'left')}, {(4, 'left'), (5, 'down'), (3, 'down')}, {(3, 'down'), (0, 'right'), (5, 'left')}, {(5, 'down'), (0, 'down'), (3, 'down')}, {(4, 'left'), (3, 'down'), (0, 'stay')}, {(4, 'down'), (0, 'stay'), (1, 'stay')}, {(4, 'down'), (0, 'stay'), (1, 'stay')}, {(4, 'down'), (0, 'stay'), (1, 'stay')}, {(4, 'left'), (0, 'stay'), (1, 'stay')}, {(4, 'down'), (0, 'stay'), (1, 'stay')}]

elapsed time: 32.10164713859558 seconds

Astar_search:

-s astar_search -c 5 -a 2 -b 1



[{(3, 'down'), (2, 'down')}, {(3, 'down'), (1, 'down')}, {(1, 'down'), (4, 'left')}, {(0, 'right'), (4, 'down')}, {(0, 'down'), (3, 'down')}, {(3, 'left'), (4, 'down')}, {(1, 'down'), (4, 'down')}, {(0, 'down'), (3, 'down')}, {(3, 'left'), (1, 'right')}, {(0, 'down'), (1, 'down')}, {(1, 'right'), (4, 'left')}, {(4, 'down'), (2, 'left')}, {(0, 'right'), (2, 'down')}, {(0, 'right'), (2, 'down')}, {(0, 'right'), (2, 'right')}, {(0, 'down'), (3, 'up')}, {(0, 'stay'), (4, 'left')}, {(4, 'left'), (2, 'down')}, {(0, 'stay'), (3, 'down')}]

elapsed time: 2.306917190551758 seconds