Austin Schaffer & John Wesley Hayhurst
Applied Artificial Intelligence 1
Homework Assignment 1
Vacuum Robots and Number Puzzles
Due September 11th, 2015

Question 2 - Simple Reflex Agent in a 2 Cell Environment

Performance Metric:
1000 available steps
1 point / step for each cleaned room

Location A	Location B	Robot Location	Performance
Dirty	Dirty	A	1999
Dirty	Dirty	B	1999
Dirty	Clean	A	2000
Dirty	Clean	B	1999
Clean	Dirty	A	1999
Clean	Dirty	B	2000
Clean	Clean	A	2000
Clean	Clean	B	2000

Overall Average: 1999.5 points

Question 3 - Simple Reflex Agent in a 2 Cell Environment with Move Penalties

Performance Metric:
1000 available steps
1 point / step for each cleaned room
-1 point for each movement

Location A	Location B	Robot Location	Performance	Sequence
Dirty	Dirty	A	1001	clean, move, clean, 997*move
Dirty	Dirty	B	1001	clean, move, clean, 997*move
Dirty	Clean	A	1001	clean, 999*move
Dirty	Clean	B	1000	move, clean, 998*move
Clean	Dirty	A	1000	move, clean, 998*move
Clean	Dirty	B	1001	clean, 999*move
Clean	Clean	A	1000	1000*move

Clean	Clean	B	1000	1000*move

Overall Average: 1000.5 points

Can a Simple Reflex Agent be perfectly rational for this environment?

No, a SRA cannot be rational for this type of environment. A SRA does not keep records, so it can only make decisions about what its sensors are currently processing. This prevents the SRA from having the ability to power down once it has explored and cleaned all of the locations in the environment.

What about an agent with state?

A model-based reflex agent has the capability to record data about the environment and would be able to outperform a SRA. It could take action to avoid repetitive motion and determine when there are no more rooms left to clean. Such an agent would have to keep track of the coordinates of every location it visits, keep track of the connections between all of the rooms it visits, and be able to generate a tree structure from that data. That way, it could perform a depth first search using only the data from its sensors.

What if the agent knew everything about the environment?

If the agent knew the coordinates, status, and cleanliness of every location in the environment, then it could be much more effective than any reflex agent. Such an agent could determine an efficient queue of actions that would take the robot through all of the dirtied locations before starting a cleaning cycle.

Question 4 - Vacuum Agents in a Square, 25-Cell Environment

Can a Simple Reflex Agent be perfectly rational for this environment?

A SRA cannot act perfectly rationally in this environment. A SRA would have no knowledge of the environment and would appear to mindlessly stumble around as it cleaned. Also, a SRA in an environment with more than 2 locations would need a randomized function to pick the next direction of travel. There is no longer 1:1 relationship between the location and the next move, so the SRA will waste a lot of time by wandering.

Would a Randomized Agent perform better than a SRA?

I am not sure what this question is asking. Any agent that is a pure SRA would have to have a randomized movement function. I created 2 different randomized SRA's for this question. Both agents perform randomized action sequences, but Agent 1 will stop to clean a dirty location. I compared both agents across multiple environments and the agent that stopped to clean the floors consistently outperformed the purely random agent.

```
Environment 1:

{ 0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1}
{ 0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1}
Starting Location: {0,  1}

Scores averaged across 10,000 trials
Semi Random Agent: 49,734
Pure Random Agent: 43,090
```

```
Environment 2:

{ 0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1,  0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1,  0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1, -1,  1,  0, -1,  0,  0,  0,  1}
Starting Location: {0,  1}

Scores averaged across 10,000 trials
Semi Random Agent: 47,452
Pure Random Agent: 41,652
```

```
Environment 3:

{ 0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1,  0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1,  0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1, -1,  1,  0, -1,  0,  0,  0,  1}
{ 0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1,  0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1,  0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1, -1,  1,  0, -1,  0,  0,  0,  1}
{ 0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1,  0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1,  0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1, -1,  1,  0, -1,  0,  0,  0,  1}
Starting Location: {0,  1}
```

```
Scores averaged across 10,000 trials
Semi Random Agent: 127,801
Pure Random Agent: 110,227
```

Design a 5x5 environment in which the Randomized Agent will perform poorly.

```
Example Poor-Performance 5x5 Environment:

{ 1, -1,  1,  1,  1}
{-1,  1,  1,  1,  1}
{ 1,  1,  1,  1,  1}
{ 1,  1,  1,  1,  1}
{ 1,  1,  1,  1,  1}
Starting Location: {0,  0}

Scores averaged across 10,000 trials
Semi Random Agent: 1
Pure Random Agent: 328
```

In order to get the Purely-Random Agent (PRAND) to outperform the Semi-Random Agent (SRAND), I had to wall both agents into a 1x1 space within a 5x5 environment.

At every t_0, the SRAND cleaned the location that it was standing on. This single clean square added 1000 to the SRAND's score after 1000 time steps. At every $t_{n>0}$, the SRAND tried to move to a new location, which subtracted 999 from its final score after 1000 time steps.

At every t_n, the PRAND picked a random action. Eventually, at time t_a, the PRAND chose to vacuum the location where it was trapped. This secured a positive score of 1000 - t_a for the PRAND after the remaining time had elapsed. After that, the PRAND continued to choose random actions. Since there are 4 possible movement actions and 2 possible stationary actions, there was a ⅓ chance that the PRAND would not lose a point at each time step. Given the scoring mechanism, this caused the PRAND to outperform the SRAND. If the scoring included a line that caused agents to lose 2 or more points for vacuuming a clean floor, then the PRAND would not outscore the SRAND in this bounded environment.

How does an Agent with State compare to the SRAND and PRAND?

```
Environment 1:

{ 0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1}
{ 0,  0,  1,  1,  0,  0,  1,  1}
{ 0,  1,  0,  1, -1,  1,  0, -1}
{ 0,  0,  0,  1,  0,  1,  0,  1}
{-1,  1,  0, -1,  0,  0,  0,  1}
Starting Location: {0,  1}
```

```
Agent With State:  49,727
Semi Random Agent: 49,736
Pure Random Agent: 43,070
```

```
Environment 2: A Porous Spiral

{ 1, -1,  1,  1,  1,  1,  1,  1,  1,  1,  1}
{ 1, -1,  1, -1, -1, -1, -1, -1, -1, -1,  1}
{ 1,  1,  1, -1,  1,  1,  1,  1,  1,  1,  1}
{ 1, -1,  1,  1,  1, -1, -1, -1,  1, -1,  1}
{ 1, -1,  1, -1,  1,  1,  1, -1,  1, -1,  1}
{ 1, -1,  1, -1, -1, -1,  1, -1,  1, -1,  1}
{ 1, -1,  1,  1,  1,  1,  1,  1,  1, -1,  1}
{ 1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1}
{ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1}
Starting Location: {0,  0}

Agent With State:  30,264
Semi Random Agent: 30,375
Pure Random Agent: 19,484
```

The simulations for these environments were run for 10,000 trials. The AWS and SRAND performed at about the same level while the PRAND was in last place. It seems that remembering where walls are located did not provide the agent with any advantage. Given that the agent only has access to tiny amounts of new information at each time step, an effective, efficient, and rational agent would need more elaborate data structures and more insightful decision making. I tried to implement a more complicated system of algorithms and data structures, but it failed to work correctly and it had to be scrapped for the current system. I honestly feel ashamed that this simple addition took so much time to implement and that I thought it would offer the agent a measurable advantage.

Question 5 - Consider the 8 and 15 Puzzles

Assume an initial state is given in csv txt file with single row with 0 for blank

1,2,3,4,5,6,0,7,8 →

1	**2**	**3**
4	**5**	**6**
0	**7**	**8**

How large is the state space for the 8-puzzle?

The state space is 9!. This is because the state space will encompass all possible configurations of the puzzle. The puzzle has 9 different nodes that it can move making the state space all possible combinations of 9 nodes, or 9!

How large is the reachable state space?

The reachable state space of the 8-puzzle is 9!/2. This is because there are some configurations that the initial starting point cannot reach. This makes the state space to be split into two different parts where the initial state is and how it can reach its goals. On the other half it is not able to reach the goals from the initial state making it another half of total state space.

<u>Is there an exemplar of an unreachable state, how large is the reachable state space from that goal?</u>

If the reachable state space is 9!/2 then something that is out of the scope would be 9!/2 + 1. Thus making the reachable state space from that goal being 9!/2 as well because it will encompass the remainder of the total state space.

<u>Is there an overlap between the reachable sets?</u>

There is no overlap between the two reachable state spaces because each reachable state space encompasses one half of the entire state space. This means no matter what using two goals within two different state spaces the encompassing state spaces will not overlap because they will fall out of scope of their reachable state space.

Question 6 - Implement the 8 Puzzle

<u>Compare best, worst, and average statistics for total path length, number of expansions during search, and maximum queue size (i.e. memory) for 12 cases randomly drawn from the reachable state space.</u>

Starting Configuration: 1,2,3,4,5,6,7,0,8			
	Cost	**Expansions**	**Queue Size**
BFSearch	1	2	6
DFSearch	29	29	27
A* Search	1	1	2
Recursive BFSearch	1	2	5

Starting Configuration: 1,2,3,4,5,6,7,8,0			
	Cost	**Expansions**	**Queue Size**
BFSearch	2	7	14
DFSearch	2	2	3
A* Search	2	2	1
Recursive BFSearch	2	6	8

Starting Configuration: 1,2,3,4,5,0,7,8,6			
	Cost	**Expansions**	**Queue Size**
BFSearch	3	22	42
DFSearch	27	27	26
A* Search	3	3	1
Recursive BFSearch	3	12	14

Starting Configuration: 1,2,0,4,5,3,7,8,6			
	Cost	**Expansions**	**Queue Size**
BFSearch	4	35	66
DFSearch	106332	115909	69557
A* Search	4	4	1
Recursive BFSearch	4	16	18

Starting Configuration: 1,0,2,4,5,3,7,8,6			
	Cost	**Expansions**	**Queue Size**
BFSearch	5	254	450
DFSearch	29	29	27
A* Search	5	5	1
Recursive BFSearch	5	54	44

Starting Configuration: 1,5,2,4,0,3,7,8,6			
	Cost	**Expansions**	**Queue Size**
BFSearch	6	555	1026
DFSearch	1794	1831	1359
A* Search	6	6	1
Recursive BFSearch	6	78	68

Starting Configuration: 1,5,2,0,4,3,7,8,6			
	Cost	**Expansions**	**Queue Size**
BFSearch	7	1726	3074
DFSearch	107179	117144	70004
A* Search	7	7	1
Recursive BFSearch	7	154	120

Starting Configuration: 1,5,2,7,4,3,0,8,6			
	Cost	**Expansions**	**Queue Size**
BFSearch	9	13502	24066
DFSearch	1793	1831	1359
A* Search	9	9	1
Recursive BFSearch	9	470	358

Starting Configuration: 1,5,2,7,4,3,8,6,0			
	Cost	**Expansions**	**Queue Size**
BFSearch	10	32227	56834
DFSearch	22580	23154	16721
A* Search	10	10	1
Recursive BFSearch	10	820	588

Starting Configuration: 1,5,2,7,4,0,8,6,3			
	Cost	**Expansions**	**Queue Size**
BFSearch	11	95422	171522
DFSearch	107179	117144	70004
A* Search	11	11	1
Recursive BFSearch	11	1306	994

Starting Configuration: 1,5,2,7,4,0,8,6,3			
	Cost	**Expansions**	**Queue Size**
BFSearch	12	544811	957954
DFSearch	70012	73154	48821
A* Search	12	89	20
Recursive BFSearch	12	3398	2352

This matches the theoretical limits described in class. A* performing the best with a tight Heuristic. The Heuristic has also been proven to behave the best for solving the N-puzzle. The Next best performing is RBFS allowing less expansions and also reducing the Queue Size. The Next best performing search was the Breadth First Search performing roughly the same as RBFS. The worst performing search was the Depth First Search. This is because the search uses a stack and exhausts all child nodes before moving onto the next nodes in the stack. There are some cases where DFS is able to perform relatively well and find the optimal path with the least cost. This is because of how the child nodes are spawned and added to the stack.

Question 7 - A* with a New Heuristic

If the heuristic overestimates for h no more than c. Meaning that if the heuristic at most adds 1 to the heuristic cost of a random node then the fcost of that node will be off by 1. If there is another node that has the same cost as the node with the faulty heuristic then the search will take one of the two nodes of the same value. When it takes the non correct, non faulty heuristic path, the cost will then be offset by that value as it will be off by one place. This will happen with increasing values of c making at the most the cost offset by c. This is not always the case as the function might choose the better path depending how it breaks the heuristic cost tie.

See the faultyManhattanDistance() function to see a heuristic cost with a value added randomly. The function will print out a confirmation that a value that is added to a random board state. One sample is when 40 random states overestimated by adding 1 to those states. This makes the c become 40 and the following was the result of the search with the starting state of 1,2,3,4,0,6,7,5,8

```
The Cost to find the solution was: 26
The Total number of expansions were: 197
The maximum queue size was: 22
```