Colin Nies & Eric Zimmerman
Introduction to Artificial Intelligence
Assignment 1: Fast Trajectory Replanning

Part 0:

All 101 x 101 grids are located in the grids folder and named grid1, grid2, …, grid50. They can be accessed using the loadFromFile(String filename) method in the grid class. I made an all-in-one method moveAgentToTarget(…) whose parameters allow you to specify the grid, the version of repeated A* you wish to use and the coordinates of the agent and target. See comments in the code above the method for more information.

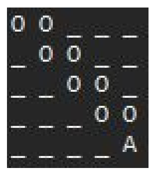
Part 1:

a) The first move of the agent is to the East because the Eastern cell f-value is lower than any other neighboring cells, making it the priority on the queue. Since it hasn't explored any other paths and does not initially know which cells are blocked, the f-value is the same as the heuristic value, so the smaller heuristic will become the priority aka to the East. The first step of A* is to go through the shortest presumed unblocked path and it knows the cell to the East is unblocked because of the original knowledge about the environment.
b) In the worst case scenario of A* in an finite gridworld, the algorithm has to visit all unblocked nodes (presuming there is not any unblocked nodes surrounded by blocked nodes, which could also be considered blocked). Either A* cannot find the target, in which case the open list is empty and every node has been visited & sorted, OR the target has been visited & added to the open list (therefore a path has been found), therefore every node has been visited, in the worst case. A* does not need to visit every cell to reach this situation however. If the target is surrounded by blocked cells, A* will get closest path possible then start to circle around the goal, looking for an opening. It would be illogical for the algorithm to explore every further unblocked cell since in pseudocode A* picks which through $f(s) \leq f(goal) = g(goal)$. A* will go through the unblocked cells that attain this goal, $g(goal)= f(goal) \geq f(s)$ once all of them are checked. A* knows there is no possible path to the target, it does not need to visit the rest of the cells. The number of expanded cells bounds the number of moves, which is bounded by the max number of expanded cells x max number of cells in a single expansion. Worst case is n nodes have to be expanded n times (n is number of unblocked nodes, each expanded once). Therefore, there is n^2 maximum moves.

Part 2:

I performed both versions of repeated forward A* on all 50 grids with starting positions (0,3) for the agent and (99,100) for the target. I keeping the position of the agent and target constant caused the agent to have no path to the target in 11 of the grids. In the other grids, the version where a smaller g-value breaks ties performed better. The agent made less moves on 27 of the 39 grids making it 69.2% better than the bigger g-value version. It performed the best on grid 33 making 80 less moves. On the other hand it did the worst on grid 3 making 72 more moves.

I also retried all 100 tests with new starting positions of (0,99) for the agent and (99,0) for the target. The version that breaks ties with smaller g values performed even better this time around. There were only 4 situations with no path to target this time. The small g version performed better on 41 out of the 47 grids making it 87.2% better in this setup. Given the data from all these tests, repeated forward A* that breaks ties with smaller g-values performs 78.2% better on average. I think the reason that this version performs better is because it traverses the grid to move towards the target in a straight line. It will attempt to move one space horizontally then one space vertically and so on. The other version will move the agent all the vertical distance first, then all the horizontal distance. I have included two pictures of how the two versions work. The left breaks ties with smaller g-values and the right version breaks ties with larger g-values. Breaking ties with smaller g-values seems to move the agent to the target "in a straight line" so to speak, it seems much more direct so in a grid world with lots of blocked cells this can be very beneficial.



All the data for part 2

Repeated Forward A* breaking ties with larger G values vs. breaking ties with lower G values. Agent at (0,3) and target at (99,100)				
**(-1 indicates no path to target)				
Bigger G breaks Ties		Smaller G Breaks Ties		
Grid	Moves	Grid	Moves	difference
1	277	1	249	28
2	-1	2	-1	0
3	229	3	301	-72
4	249	4	227	22
5	273	5	255	18
6	245	6	251	-6
7	245	7	249	-4
8	269	8	263	6
9	-1	9	-1	0
10	-1	10	-1	0
11	259	11	251	8
12	249	12	251	-2
13	-1	13	-1	0
14	253	14	241	12
15	253	15	241	12
16	261	16	261	0
17	233	17	223	10
18	267	18	213	54
19	309	19	279	30
20	-1	20	-1	0
21	249	21	231	18
22	307	22	297	10
23	295	23	279	16
24	251	24	235	16

25	259	25	221	38
26	263	26	209	54
27	263	27	245	18
28	-1	28	-1	0
29	237	29	245	-8
30	267	30	231	36
31	229	31	229	0
32	231	32	219	12
33	333	33	253	80
34	283	34	259	24
35	-1	35	-1	0
36	263	36	267	-4
37	255	37	287	-32
38	247	38	259	-12
39	271	39	259	12
40	-1	40	-1	0
41	243	41	267	-24
42	259	42	223	36
43	313	43	239	74
44	241	44	245	-4
45	283	45	245	38
46	241	46	247	-6
47	293	47	263	30
48	-1	48	-1	0
49	245	49	247	-2
50	237	50	273	-36

Repeated Forward A* breaking ties with larger G values vs. breaking ties with lower G values. Agent at (0,99) and target at (99,0)				
**(-1 indicates no path to target)				
Bigger G breaks Ties		Smaller G Breaks Ties		
Grid	Moves	Grid	Moves	difference
1	255	1	257	-2
2	291	2	251	40
3	249	3	247	2
4	279	4	211	68
5	265	5	253	12
6	-1	6	-1	0
7	271	7	235	36
8	235	8	227	8
9	257	9	247	10
10	247	10	259	-12
11	287	11	241	46
12	-1	12	-1	0
13	-1	13	-1	0
14	239	14	219	20
15	275	15	249	26
16	275	16	243	32
17	259	17	261	-2
18	295	18	287	8
19	269	19	269	0
20	261	20	239	22
21	295	21	267	28
22	277	22	255	22
23	309	23	233	76
24	319	24	227	92
25	283	25	247	36
26	295	26	223	72
27	291	27	263	28
28	281	28	261	20

29	267	29	265	2
30	303	30	253	50
31	257	31	233	24
32	299	32	289	10
33	277	33	257	20
34	241	34	231	10
35	263	35	295	-32
36	277	36	271	6
37	259	37	253	6
38	259	38	275	-16
39	235	39	213	22
40	259	40	235	24
41	321	41	263	58
42	245	42	239	6
43	279	43	257	22
44	325	44	255	70
45	249	45	247	2
46	305	46	267	38
47	275	47	237	38
48	271	48	299	-28
49	299	49	289	10
50	283	50	277	6

Part 3:

I performed the forward and backward A* on all 50 grids with starting positions (0,3) for the agent and (99,100) for the target. In this scenario, the forward version performed much better. The forward version made less moves in 35 out of the 39 grids with paths. On grid 3, the backward version made a whopping 157 extra moves.

I also performed forward and backward A* on all 50 grids with starting positions (0,99) for the agent and (99,0) for the target. This time, the forward version made less moves in 43 of the 47 grids with paths. Combining the results from all these test, repeated forward a star performs better 91.0% of the time. I think the reason the backward version causes the agent to make so many moves is because it doesn't add the cells to the path that the agent has knowledge about until the end. It will find a very direct path in the unknown cell region but once it looks at cells the agent knows about it can have to make a lot of twists and turns to get past the blocked cells to the agent. Whereas every time the path is computed in the forward version, known and unblocked cells are always added to the shortest presumed unblocked path first.

All the data for Part 3

Repeated Forward A* vs. Repeated backward A* with agent at (0,3) and target at (99,100)				
**(-1 indicates no path to target)				
forward		backward		
Grid	Moves	Grid	Moves	difference
1	277	1	318	-41
2	-1	2	-1	0
3	229	3	386	-157
4	249	4	325	-76
5	273	5	387	-114
6	245	6	315	-70

7	245	7	299	-54
8	269	8	320	-51
9	-1	9	-1	0
10	-1	10	-1	0
11	259	11	324	-65
12	249	12	290	-41
13	-1	13	-1	0
14	253	14	344	-91
15	253	15	276	-23
16	261	16	314	-53
17	233	17	362	-129
18	267	18	343	-76
19	309	19	348	-39
20	-1	20	-1	0
21	249	21	322	-73
22	307	22	333	-26
23	295	23	267	28
24	251	24	353	-102
25	259	25	334	-75
26	263	26	355	-92
27	263	27	342	-79
28	-1	28	-1	0
29	237	29	300	-63
30	267	30	284	-17
31	229	31	284	-55
32	231	32	338	-107
33	333	33	325	8
34	283	34	297	-14
35	-1	35	-1	0
36	263	36	260	3
37	255	37	296	-41
38	247	38	263	-16
39	271	39	284	-13
40	-1	40	-1	0
41	243	41	374	-131
42	259	42	313	-54
43	313	43	305	8
44	241	44	295	-54
45	283	45	325	-42
46	241	46	328	-87
47	293	47	384	-91
48	-1	48	-1	0
49	245	49	271	-26
50	237	50	335	-98

Repeated Forward A* vs. Repeated backward A* with agent at (0,99) and target at (99,0)				
**(-1 indicates no path to target)				
forward		backward		
Grid	Moves	Grid	Moves	difference
1	255	1	285	-30
2	291	2	342	-51
3	249	3	312	-63
4	279	4	281	-2
5	265	5	326	-61
6	-1	6	-1	0
7	271	7	366	-95
8	235	8	285	-50
9	257	9	313	-56
10	247	10	319	-72

11	287	11	289	-2
12	-1	12	-1	0
13	-1	13	-1	0
14	239	14	328	-89
15	275	15	404	-129
16	275	16	319	-44
17	259	17	340	-81
18	295	18	282	13
19	269	19	313	-44
20	261	20	328	-67
21	295	21	309	-14
22	277	22	350	-73
23	309	23	315	-6
24	319	24	314	5
25	283	25	292	-9
26	295	26	257	38
27	291	27	291	0
28	281	28	277	4
29	267	29	336	-69
30	303	30	311	-8
31	257	31	302	-45
32	299	32	400	-101
33	277	33	310	-33
34	241	34	303	-62
35	263	35	321	-58
36	277	36	318	-41
37	259	37	298	-39
38	259	38	335	-76
39	235	39	306	-71
40	259	40	404	-145
41	321	41	321	0
42	245	42	310	-65
43	279	43	375	-96
44	325	44	375	-50
45	249	45	294	-45
46	305	46	304	1
47	275	47	293	-18
48	271	48	309	-38
49	299	49	322	-23
50	283	50	298	-15

Part 4:

a) Manhattan distances are consistent in gridworlds, moving in only 4 directions because they are found by summing the vertical and horizontal distances. Since A* can only move in compass directions, to get to a certain point the distance will be the same, no matter what combination of horizontal and vertical steps. If A* could go diagonal, Manhattan distances would be a greater value and not consistent. Going diagonal would allow A* to go to a diagonal cell in 1 move that would have previously taken 2 moves to get to.

b) First, we have to prove Adaptive A* h-values, $h_{new}(c)$ are consistent. Consider h(c) as the Manhattan distance values and $h_{new}(c)$ as the difference between g(goal) and g(c). c will be the cell and c' the successor cell, the cost to move from c to c' will be one. We already proved that Manhattan distances are consistent in gridworlds - h(c) ≤ h(c') + *cost*(c,a,c'), this holds for h(s) but we want to prove $h_{new}(c) \leq h_{new}(c') + cost(c,a,c')$. We can substitute $h_{new}(c)$ = g(goal) - g(c') and $h_{new}(c')$ = g(goal) - g(c') → g(goal) -

$g(c) \leq g(goal) - g(c') + cost(c,a,c')$. We can then simplify out g(goal) → $-g(c) \leq -g(c') + cost(c,a,c')$ then switch our inequality → $g(c) \geq g(c') - cost(c,a,c')$. From this equation, you can see for all 4 compass directions, it stays true. If g(c') is bigger than g(c), subtract by one will make them equal (because their distance from each other is always 1). If g(c') is smaller than g(c) subtracting 1 will make it even smaller. The triangle inequality is satisfied and $h_{new}(c)$ values are consistent. Now, we have to prove these h-values remain consistent if action costs increase. Looking at the previous inequality, $h_{new}(c) \leq h_{new}(c') + cost(c,a,c')$. If we have an action cost increase → *cost*- action cost before increase, *cost'*- action cost after increase. Then, $h_{new}(c) \leq h_{new}(c') + cost(c,a,c') \leq h_{new}(c') + cost(c,a,c')$. So, the inequality remains consistent proving Adaptive A* leaves initially consistent h-values consistent even if action costs increase.

Part 6:

I can imagine a couple of ways we can decrease memory usage. I have a method in the Tree class called getPath(…) that returns the path to goal in the form of a stack. I realize now that this is redundant because I could of just returned the goal pointer and followed parent pointers to start state. My grid is represented as a 2D array of Cell objects. Each cell has 5 ints x, y, h, g, f and one Boolean isBlocked. My tree has pointers to the Cell objects in the grid so it only takes up space for each reference. In the Cell class I could dynamically make x,y and h short ints in the cases where the grid dimensions aren't extremely huge. I could also change the way my Tree works by removing all child pointers for each node and just keeping parent pointers. Additionally, now that I think about it I could just make the parent pointer a field in the Cell class. The way I represent the agent's knowledge of cells could also be improved at least for the non-adaptive versions of repeated A*. I represent the cells that the agent knows about with a 2D Boolean array of the same dimensions as the grid but I suppose I could embed this info in each cell object. So in the best case scenario the cell class would have 3 short ints(x, y, h), 2 regular ints (g, f) 2 booleans (isBlocked, agentKnows) and one 2-bit reference to parent. If we assume shorts are 2 bytes and Boolean 1 bit then one cell object is 14 bytes and 4 extra bits or just 116 bits. A grid of size 1001 x 1001 would be 116,232,116 bits (~14,403,764 bytes). The biggest grid we can generate within 4MB is 537 x 537. 537 x 537 x 116 = 33,450,804 bits, 1 MB = 4,194,304 bytes = 33,554,432 bits