COMP8620 Assignment 1 Report: Trajectory Replanning for Computer Games

Bodhi Philpot (u4004519)

September 2008

1 Data Structures

Each square of the game world is represented by a Java class `State`. These states are only created as needed – if, for example the point $(0, 0)$ is never accessed by any of the repeated searches, it is never created. Each state contains g, h and *search* values, and a pointer to its parent state.

States contain their (x, y) coordinates even though they are usually accessed via coordinate lookups, as this simplified construction of the path-finding by following the parent pointers to find the next path step, thus avoiding having to determine the (x, y) coordinate for each state.

The states are managed by a `Maze` class, that loads in the text descriptions of mazes, and stores the start and goal points in the maze. It also keeps track of the agent's knowledge of the world by storing a hidden complete map of the maze along with an incomplete map used by the agent to search for paths through the maze. As the agent moves about, its copy of the maze is updated with knowledge of obstructions in the game world from the hidden map.

A `Search` class implements a simple A* algorithm with optional updating of $h(s)$ according to the rules given for adaptive A*. This class performs a single A* search on the persistent maze. The search is not limited to searching from the maze start to the maze goal, but will find a path from two arbitrary points. This allows for Repeated Backward A* to search from the maze goal *back* to the current agent location.

The overall path finding is conducted by a `PathFinder` class – implementing Repeated Forward A* – with subclasses to implement Repeated Backward A* (`RepeatedBackwardPathFinder`) and Adaptive A* (`AdaptivePathFinder`). Repeated Backward A* needs to reset $h(s)$ for each state in the maze before searching again, and needs to reverse the found path before moving the agent. The only implementation difference between Adaptive A* and Repeated Forward A* is that Adaptive A* enables the heuristic updating on its searches.

The main class for the program is `Main`. Usage is
`java -cp . Main <maze filename> <search type>`
with search type of 1: Backward A*; 2: Adaptive A*; Repeated Forward A* otherwise

1.1 Tie Breaking Method

When two nodes on the open list had an equal value for $f(s)(= g(s) + h(s))$, the node with a greater $g(s)$ value was selected for expansion. When this wasn't sufficient to distinguish two nodes, the node added to the open list earlier was selected for expansion.

1.2 Visualisations

The assignment was originally written with Processing[1] – a Java-based graphical environment – to visualise the search progress. Each step the agent took was drawn on screen – along with planned path and search open and closed lists – to help understand the algorithms used in the assignment. The figures in this report are from this version. The code was then converted to a regular Java program to complete the assignment requirements. This Processing program is available in the `astar` directory, but has not been commented or otherwise made more understandable. The program will loop through each maze (from 1 to 50) choosing a random algorithm (from the three in the assignment) to use to find a path from the start to the goal.

2 Experiment Results

The raw results from running Repeated Forward A*, Repeated Backward A* and Adaptive A* on each of 50 randomly generated mazes[2]. The results can be seen in table 1. Normalising these results to the performance of Repeated Forward A* (see table 2), the results can be compared more easily across different mazes.

Averaging the normalised values to get a simple overview of performance, Repeated Backwards A* generated paths that were 27% longer, and expanded *12 times* as many nodes as Repeated Forward A*. Adaptive A* performed much better, generating paths that were only 1% longer on average, while only generating 78% as many nodes as Repeated Forward A*.

[1] `http://processing.org`
[2] Generated with the provided program

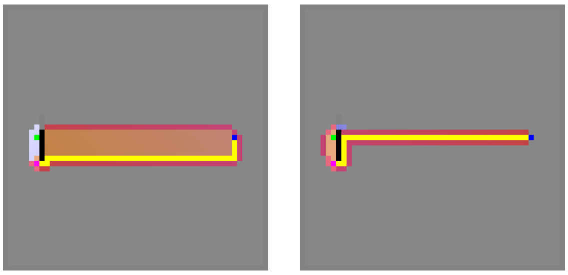

Figure 1: Repeated Backward A* (left) compared with Repeated Forward A* (right). As Repeated Backward A* searches from an open area near the goal (blue square) into a closed area around the agent (pink square), it expands more nodes (orange) than Repeated Forward A*. The green square is the Agent's origin.

2.1 Repeated Forward A*

Repeated Forward A* proved to be effective at finding short paths through each maze.

2.2 Repeated Backward A*

Repeated Backward A* usually expanded many more nodes than did Repeated Forward A* or Adaptive A*. This is assumed to be because the general case in this type of maze – where the area around the maze goal (search origin) is relatively free of obstructions but the area around the agent (search target) is heavily obstructed – is a pathologically bad case for an optimal A* type of search. An extreme case of this is shown in figure 1. Using an A* variation such as weighted A* would make this problem less severe, with the cost of not generating optimal path plans.

Another overhead of Repeated Backward A* is the resetting of $h(s) = 0$ for each state whenever the current path is found to be blocked (resulting in a new A* search). As the search goal – the agent – is moving, the heuristics cannot be reused between searches.

2.3 Adaptive A*

Because of the adjustment to $h(s)$, Adaptive A* would occasionally make different path decisions compared to Repeated Forward A*, an example of which can be seen in figure 2. Adaptive A* explored the top–left are of the maze more than Repeated Forward A*, which gave up earlier. The main area of path difference can be seen in figure 3. This is because as Adaptive A* approached the point of divergence, the multiple path re-planning caused $h(s)$ for the shorter path to be increased enough such that Adaptive A* predicted the other path direction to be shorter.

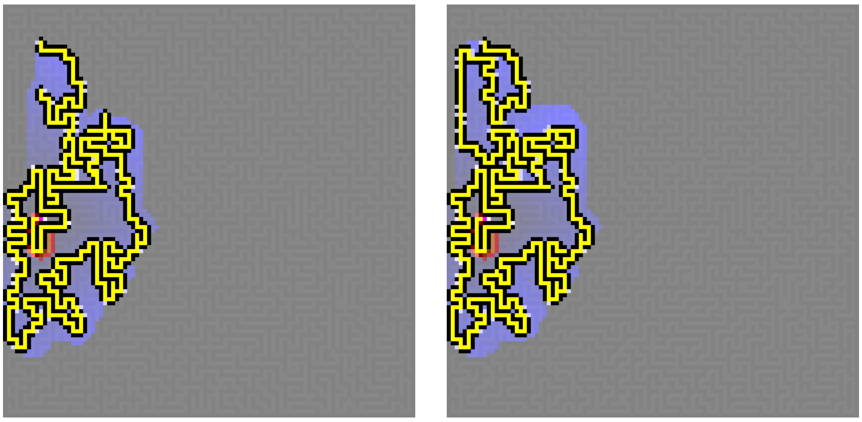

Figure 2: Solution of Maze 1 by Repeated Forward A* (left) and Adaptive A* (right)

Figure 3: Zoomed section of figure 2, focusing on point of divergence of paths followed by Repeated Forward A* (left) and Adaptive A* (right)

However, even though the path generated by Adaptive A* was longer (618 vs. 584), Adaptive A* expanded 346 (4%) *fewer* nodes in total than Repeated Forward A*.

2.4 Actual Results

In the following tables, R.FA* for Repeated Forward A*; R.BA* Repeated Backward A* and Adap. A* for Adaptive A*.

Table 1: Raw Search Performance Results

Maze	Path Length			Expanded Nodes		
	R.FA*	R.BA*	Adap. A*	R.FA*	R.BA*	Adap. A*
1	584	686	618	10262	55443	9916
2	400	2408	404	7323	1225407	6610
3	406	408	406	13010	47069	9559
4	54	104	54	409	1355	381
5	496	680	774	15455	169817	19029
6	2166	1940	2296	139799	1326396	98337
7	1526	1280	1202	65742	243671	28518
8	1832	1982	1892	262140	1396567	86242
9	1284	1396	1268	251888	290866	76377
10	234	172	256	10419	7538	5681
11	888	1178	826	36281	388156	31765
12	870	818	854	34714	167784	26065
13	710	592	700	31047	149504	21113
14	696	1266	708	16369	509647	14987
15	536	298	536	14098	86377	13256
16	902	1562	904	27266	418702	24977
17	1234	1272	1290	87848	201342	37931
18	2078	1784	2192	145632	432279	65136
19	738	1204	698	64909	333094	32430
20	404	350	404	9514	34258	6918
21	976	1054	1052	33351	525160	31523
22	488	458	488	6360	65404	5579
23	576	650	578	16042	163782	14676
24	1040	1630	1028	35055	689700	29559
25	582	332	714	25479	38497	36314
26	576	508	592	10462	33472	8958
27	2008	1306	1996	130185	546826	70497

28	50	50	50	316	499	299
29	1662	1652	980	67536	585594	19665
30	1144	1384	1140	35941	593283	32012
31	454	1846	440	11068	527248	9324
32	1128	1826	1128	87719	989497	56306
33	548	554	548	52634	49844	28753
34	910	744	910	26234	288149	24199
35	780	1404	782	31814	1112966	30611
36	1014	1046	998	26120	260590	24223
37	678	458	678	15296	31963	11719
38	546	940	546	15092	200604	14724
39	296	358	296	7742	27453	7256
40	918	350	480	37409	36241	7578
41	780	776	774	24320	63105	15869
42	996	1288	1442	46745	1219643	69172
43	272	280	272	9627	36101	7089
44	676	706	676	17319	140786	15588
45	450	416	450	17204	134940	14135
46	1964	842	2428	543870	321799	258914
47	852	986	884	184050	265938	91968
48	480	504	480	12674	74208	11451
49	1404	2136	1478	53317	1074798	49254
50	564	1244	588	15728	234296	17286

Table 2: Normalised Performance Results

Maze	Path Length			Expanded Nodes		
	R.FA*	R.BA*	Adap. A*	R.FA*	R.BA*	Adap. A*
1	1.00	1.17	1.06	1.00	5.40	0.97
2	1.00	6.02	1.01	1.00	167.34	0.90
3	1.00	1.00	1.00	1.00	3.62	0.73
4	1.00	1.93	1.00	1.00	3.31	0.93
5	1.00	1.37	1.56	1.00	10.99	1.23
6	1.00	0.90	1.06	1.00	9.49	0.70
7	1.00	0.84	0.79	1.00	3.71	0.43
8	1.00	1.08	1.03	1.00	5.33	0.33
9	1.00	1.09	0.99	1.00	1.15	0.30
10	1.00	0.74	1.09	1.00	0.72	0.55
11	1.00	1.33	0.93	1.00	10.70	0.88
12	1.00	0.94	0.98	1.00	4.83	0.75
13	1.00	0.83	0.99	1.00	4.82	0.68
14	1.00	1.82	1.02	1.00	31.13	0.92
15	1.00	0.56	1.00	1.00	6.13	0.94
16	1.00	1.73	1.00	1.00	15.36	0.92
17	1.00	1.03	1.05	1.00	2.29	0.43
18	1.00	0.86	1.05	1.00	2.97	0.45
19	1.00	1.63	0.95	1.00	5.13	0.50
20	1.00	0.87	1.00	1.00	3.60	0.73
21	1.00	1.08	1.08	1.00	15.75	0.95
22	1.00	0.94	1.00	1.00	10.28	0.88
23	1.00	1.13	1.00	1.00	10.21	0.91
24	1.00	1.57	0.99	1.00	19.67	0.84
25	1.00	0.57	1.23	1.00	1.51	1.43
26	1.00	0.88	1.03	1.00	3.20	0.86
27	1.00	0.65	0.99	1.00	4.20	0.54
28	1.00	1.00	1.00	1.00	1.58	0.95
29	1.00	0.99	0.59	1.00	8.67	0.29
30	1.00	1.21	1.00	1.00	16.51	0.89
31	1.00	4.07	0.97	1.00	47.64	0.84
32	1.00	1.62	1.00	1.00	11.28	0.64
33	1.00	1.01	1.00	1.00	0.95	0.55
34	1.00	0.82	1.00	1.00	10.98	0.92
35	1.00	1.80	1.00	1.00	34.98	0.96
36	1.00	1.03	0.98	1.00	9.98	0.93

37	1.00	0.68	1.00	1.00	2.09	0.77
38	1.00	1.72	1.00	1.00	13.29	0.98
39	1.00	1.21	1.00	1.00	3.55	0.94
40	1.00	0.38	0.52	1.00	0.97	0.20
41	1.00	0.99	0.99	1.00	2.59	0.65
42	1.00	1.29	1.45	1.00	26.09	1.48
43	1.00	1.03	1.00	1.00	3.75	0.74
44	1.00	1.04	1.00	1.00	8.13	0.90
45	1.00	0.92	1.00	1.00	7.84	0.82
46	1.00	0.43	1.24	1.00	0.59	0.48
47	1.00	1.16	1.04	1.00	1.44	0.50
48	1.00	1.05	1.00	1.00	5.86	0.90
49	1.00	1.52	1.05	1.00	20.16	0.92
50	1.00	2.21	1.04	1.00	14.90	1.10