02285 - Assignment 1

Emil Rask Maagaard - s142955

February 10, 2015

Introduction

In the process of making this assignment I have been having discussions with Bjarke Vad Andersen (s142941).

Exercise 1 (Search strategies)

The initial search client benchmark

Level	Client	Time	Memory used	Solution length	Nodes explored
SAD1	BFS	0.07 s	11.52 MB	19	78
SAD1	DFS	0.05 s	8.96 MB	27	44
SAD2	BFS	-	-	-	-
SAD2	DFS	4.57 s	509.2 MB	5781	6799
custom	BFS	-	-	-	-
custom	DFS	0.06 s	11.52 MB	45	60

SAD2 with BFS

It can seem like there are many more solutions, because the SAD2 level has 4 A's, and not just one. The BFS therefore have many more possible states to explore, which results in no solution to be found in the limited time or with the limited memory. This search was done with 8GB of RAM allocated for the SearchClient.

DFS search strategy

The DFS client is implemented using a java.Util.Stack as the frontier, as the stack behavior corresponds to that of the DFS strategy.

Custom level

The custom designed level is a derivation of SAD2, where some of the possible spaces are filled to limit the possible movements. The possible A's are also moved a little bit.

```
++++++++++++++++++
+0A    +++       +
+ A           ++++
+ A    + +       +
+             ++A+
+++++        +++a+
++++++++++++++++++
```

Exercise 2 (Optimisations)

Optimising memory use

The excessive memory use is rectified by creating temporary 2d-char-arrays to contain the level walls, boxes and goals, so the actual required size is known before the Node is created. The required size of the arrays are set in the Node constructor. By doing this only the needed amount of memory is allocated.

The walls and goals of the ChildNode are just set to the walls and goals of the parent Node, instead of calling System.arrayCopy, which is perfectly fine as they are static.

Regarding the boxes, the structure used is not very suitable for the case, since there are very few boxes compared to the possible large 2-dimensional arrays they are stored in. Even though the array structure has a very short look-up time, a hash map containing the coordinates of each box may provide a better solution. The hash map size would be limited to the number of boxes and also have very fast look-ups although not as fast as the array structure.

Benchmark with optimisations

Level	Client	Time	Memory used	Solution length	Nodes explored
SAD1	BFS	0.01 s	5.12 MB	19	78
SAD1	DFS	0.01 s	5.12 MB	27	44
SAD2	BFS	-	-	-	-
SAD2	DFS	1.23 s	23.10 MB	5781	6799
custom	BFS	-	-	-	-
custom	DFS	0.01 s	5.12 MB	45	

Exercise 3 (Heuristics)

BestFirst search strategy

The BestFirst search strategy is implemented using a java.Util.PriorityQueue as the frontier. The queue utilizes the Comparator<Node> from Heuristics to sort the queue when adding elements. In order to optimize the state exploration, an extra frontier is used, oldFrontier. When a states is polled from the frontier, the rest of the states are moved to oldFrontier and frontier is cleared. The frontier is then used to store the unexplored states of the chosen node. In case of a dead end i.e. an empty frontier, an unexplored state is polled from oldFrontier.

Heuristics function

The heuristic functions functions calculates the Pythagorean distance from the agent to the box and from the box to the goal based on the array-coordinates of each. The goal and box coordinates are found in the pre-processing. As the implementation is right now, there can be and issue with multiples of same box and goal name, because it is stored in a hash map with the box and goal character as key.

Admissibility

The heuristic function is admissible, as it calculates the Pythagorean distance between agent, box and goal, without regarding any possible obstacles. This will always be the shortest distance.

Competition levels

The SearchClient could not solve either of the competition levels. The reason for this being that the implemented algorithms doesn't take multiple and different goals and boxes into account.

Benchmark

Level	**Evaluation**	**Time**	**Memory used**	**Solution length**	**Nodes explored**
SAD1	A*	0.01 s	5.12 MB	19	35
SAD1	WA*	0.01 s	5.12 MB	19	19
SAD1	Greed	0.01 s	5.12 MB	19	19
SAD2	A*	0.01 s	5.12 MB	19	19
SAD2	WA*	0.01 s	5.12 MB	19	19
SAD2	Greedy	0.01 s	5.12 MB	19	19
custom	A*	0.01 s	5.12 MB	19	19
custom	WA*	0.01 s	5.12 MB	19	19
custom	Greedy	0.01 s	5.12 MB	19	19
FireFly	A*	-	-	-	-
FireFly	WA*	-	-	-	-
FireFly	Greedy	-	-	-	-
Crunch	A*	-	-	-	-
Crunch	WA*	-	-	-	-
Crunch	Greedy	-	-	-	-