**F29AI – Coursework 1**

**A\* Search**

**and Automated Planning**

**Edinburgh - Group CW1 - UG 3**

**Donnie Stark (H00346580)**

**Marc Crawford (H00400967)**

Table of Contents

[Part 1A 3](#_Toc180655576)

[Part 1B 4](#_Toc180655577)

[Part 2A&B 6](#_Toc180655578)

[Part 2C 7](#_Toc180655579)

# Part 1A

A diagram of a grid

Description automatically generatedA diagram of a diagram

Description automatically generated with medium confidence1.

2.

Using the manhattan distance heuristic we gave each spot in the “grids” coordinates and calculated it using the coordinates of the node and the goal node; ∣x2​−x1​∣+∣y2​−y1​∣ which gave us:

|  |  |
| --- | --- |
| Node | Heuristic |
| S | h(s) = 7 |
| A | h(s) = 6 |
| E | h(s) = 6 |
| F | h(s) = 5 |
| L | h(s) = 5 |
| M | h(s) = 4 |
| P | h(s) = 6 |
| H | h(s) = 4 |
| B | h(s) = 4 |
| I | h(s) = 3 |
| N | h(s) = 2 |
| Q | h(s) = 3 |
| C | h(s) = 3 |
| J | h(s) = 2 |
| O | h(s) = 1 |
| D | h(s) = 2 |
| R | h(s) = 2 |
| K | h(s) = 1 |
| T | h(s) = 1 |
| G | h(s) = 0 |

|  |  |
| --- | --- |
| Node | Heuristic |
| S | h(s) = 7 |
| A | h(s) = 6 |
| D | h(s) = 6 |
| E | h(s) = 5 |
| J | h(s) = 5 |
| K | h(s) = 4 |
| N | h(s) = 4 |
| O | h(s) = 3 |
| T | h(s) = 3 |
| F | h(s) = 4 |
| P | h(s) = 2 |
| B | h(s) = 4 |
| H | h(s) = 3 |
| L | h(s) = 2 |
| Q | h(s) = 1 |
| C | h(s) = 5 |
| I | h(s) = 4 |
| M | h(s) = 3 |
| R | h(s) = 2 |
| U | h(s) = 1 |
| G | h(s) = 0 |

3.

1st Grid

F = {{S, F = 0+7 = 7}} C = {}

Remove S and expand

F = {{A, F = 1+6 = 7} {E, F = 2+6 = 8}} C = {S}

Remove A and expand

F = {{E, F= 2+6 = 8}{F, F=3+5 = 8}} C = {S, A}

Remove E and expand (tie broken by alphabetical order)

F= {{F, F = 2+6 = 8}{L, F= 3+5 = 8}} C = {S, A, E}

Remove F and expand (tie broken by alphabetical order)

F = {{L, F= 3+5 = 8}{H, F = 4+4 = 8}{M, F= 4+4 = 8}} C={S, A, E, F}

Remove H and expand (tie broken by alphabetical order)

F={{L, F= 3+5 = 8}{M, F= 4+4 = 8}{I, F= 6+3 = 9}} C = {S, A, E, F, H}

…

..

.

States Expanded = {S, A, E, F, H, L, M, I, J, K, G}

Goal Path = {S, A, F, H,  I, J, K, G}

Total Cost = 9

2nd Grid

F = {{S, F= 0+7 = 7}} C={}

Remove S and expand

F= {{A, F=1+6 = 7}{D, F= 2+6 = 8}} C ={S}

Remove A and expand

F = {{D, F= 2+6 = 8}{E, F=2+5 = 7}} C ={S, A}

Remove E and expand

F = {{D, F= 2+6 = 8}{K, F = 3+4 = 7}{F, F= 3+4 = 7}} C = {S, A, E}

Remove F and expand (tie broke by alphabetical order)

F = {{D, F= 2+6 = 8}{K, F = 3+4 = 7}{H. F = 5+3 = 8}} C = {S, A, E, F}

Remove K and expand

F = {{D, F= 2+6 = 8}{H. F = 5+3 = 8}{J, F= 4+5 = 9}{P, F = 6+2 = 8}{N, F = 6+4 = 10}  C={S, A, E, F, K}

…

..

.

States Expanded = {S, A, E, F, K, O, D, H, L, P, Q}

Goal Path(2) =

1 = {S, A, E, F, H, L, Q, G}

2 = {S, A, E, K, O, P, Q, G}

Total Cost = 8

# Part 1B

To implement the A\* Search to work on the grids given above we created 3 classes; AStarSearchOrder, HeuristicState and Main. HeuristicState was created to implement state classes, allowing the program to get the heuristic for the node created. To do this we implemented providing coordinates in the grid for each node as well as the value and what the goalNode is so we can use it’s coordinates to find the Manhattan Distance Heuristic.

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generatedA black screen with text

Description automatically generated with medium confidence  
Using the method above we implemented getHeuristic() and isGoal() simply returning the true or false of whether it is the goal node.

We then implemented the SearchOrder and its methods with a class AStarSearchOrder. We created a hash set to keep track of visitedStates.  


We then implemented addToFringe(). We first implemented a priorityQueue as needed with A\* Search to order the fringe by the minimum distance + heuristic distance. If there was a tie, it then picks the state which is first in alphabetical order.

A screen shot of a computer program

Description automatically generated

We then ensure add the frontier FringeNode’s to the fringe priority queue ensuring they haven’t previously been visited.

A screen shot of a computer code

Description automatically generated

We then add the contents of the priority queue to our frontier

A screen shot of a computer code

Description automatically generated

Then in the Main class we simply create our nodes and set up our grid and all the pathways between nodes. Creating Nodes:



Creating a pathway between nodes:

A screenshot of a computer screen

Description automatically generated

And our output matched what we got by manually doing it above. (This is the second grid as an example).

A grey screen with white text

Description automatically generated

A screen shot of a computer screen

Description automatically generatedA screenshot of a computer program

Description automatically generated

# Part 2A&B

During the implementation of the domain and problems in PDDL, we have used PDDL support on VSCODE and throughout have used the BFWS - Best First Width Search planner as shown in the screenshots below.

A black background with white text

Description automatically generatedWhen setting up our domain we tried to create predicates that covered all the possible relations between types involved. For example, relationships such as uuv and location, uuv can be at a location, uuv can move between locations (which is a future action) there is paths between locations. This was the thought and planning process we went for for each type we specified. Here are some example predicates.

A screen shot of a computer screen

Description automatically generated

Our actions then also reflected all the interactions that needed to be implemented in order to be able to achieve our problem mission goals stated later on. We identified these key actions needed, Deploying a UUV, UUV moving between locations, Ship moving between locations, UUV capturing an image, UUV conducting a solar scan, UUV transmitting data back to the ship, UUV collecting a sample, UUV returning to the ship, UUV returning the sample. Examples:

A screen shot of a computer program

Description automatically generatedA screen shot of a computer program

Description automatically generated

Here are the planner outputs we got for each problem:

A screen shot of a computer

Description automatically generatedProblem 1: Problem 2:

A screen shot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generatedProblem 3:

# Part 2C

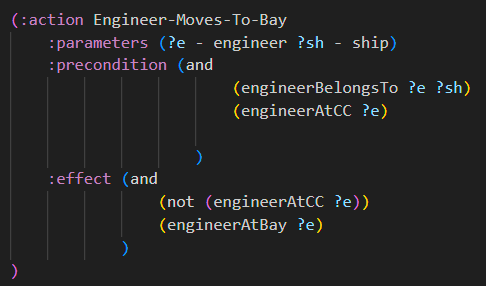
When implementing the second domain for this part, we added on to the previous domain for part 2 A&B. We added a new type, an engineer. We also then added predicates that describe our engineer:

A black background with white text

Description automatically generated

We identified the actions, Engineer moves to bay and Engineer moves to control centre:

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer

Description automatically generatedOur output for problem 4:

Our video demo link: