Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	F29AI – Artificial Intelligence and Intelligent Agents
Type of assessment:	**Group**
Coursework Title:	A* Search and Automated Planning
Student Name:	Aditya Vikram Singh Dahiya
Student ID Number:	H00444458

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* *Aditya Dahiya*

Date: *24/10/2024*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

Student Declaration of Authorship

HERIOT WATT UNIVERSITY
UK | DUBAI | MALAYSIA

Course code and name:	F29AI – Artificial Intelligence and Intelligent Agents
Type of assessment:	**Group**
Coursework Title:	A* Search and Automated Planning
Student Name:	Keerthana Babu Gopakumar Nair
Student ID Number:	H00421150

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.

- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the University's website, and that I am aware of the penalties that I will face should I not adhere to the University Regulations.

- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on Academic Integrity and Plagiarism

Student Signature *(type your name):* *Keerthana Babu Gopakumar Nair*

Date: *24/10/2024*

Copy this page and insert it into your coursework file in front of your title page.
For group assessment each group member must sign a separate form and all forms must be included with the group submission.

Your work will not be marked if a signed copy of this form is not included with your submission.

F29AI - Artificial Intelligence and Intelligent Agents

A* Search and Automated Planning

Dubai Cw 1 UG Group 20

Semester 1 - 2024/2025

24/10/2024

Aditya Vikram Singh Dahiya

H00444458

avsd2000@hw.ac.uk

Keerthana Babu Gopakumar Nair

H00421150

Kbgn2000@hw.ac.uk

Dubai Campus

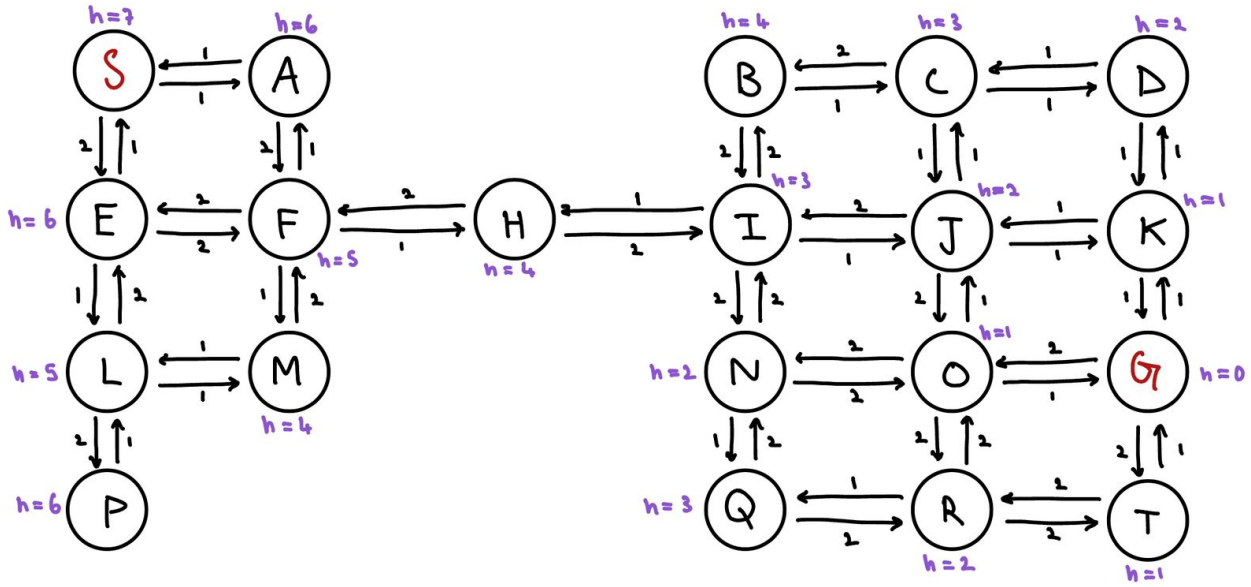
Part 1: A* Search

Part 1A (5 marks): A* Search by Hand

I. A* Search formula: $f(n) = g(n) + h(n)$, where $f(n)$ is the total estimated cost of the cheapest path through node n, $g(n)$ is actual cost from the start to the node n and $h(n)$ is the heuristic estimate from node n to the goal node.

1st Grid Environment:

Grid 1 represented as a graph:



Fringe list:	Closed Node List:
F = {(S, f =0+7= 7)}	C = { ∅ }
F = {(A, f =1+6= 7), (E, f =2+6= 8)}	C = {S}
F = {(F, f =3+5= 8), (E, f =2+6= 8)}	C = {S, A}
F = {(L, f =3+5= 8), (F, f =3+5= 8)}	C = {S, A, E}
F = {(H, f =4+4= 8), (M, f=4+4=8), (L, f =3+5= 8)}	C = {S, A, E, F}
F = {(I, f=6+3=9), (M, f=4+4=8), (L, f =3+5= 8)}	C = {S, A, E, F, H}
F = {(P, f=5+6=11), (I, f=6+3=9), (M, f=4+4=8)}	C = {S, A, E, F, H, L}
F = {(P, f=5+6=11), (I, f=6+3=9)}	C = {S, A, E, F, H, L, M}
F = {(B, f=8+4=12), (J, f=7+2=9), (N, f=8+2=10), (P, f=5+6=11)}	C = {S, A, E, F, H, L, M, I}
F = {(C, f=8+3=11), (K, f=8+1=9), (O, f=9+1=10) (B, f=8+4=12), (N, f=8+2=10), (P, f=5+6=11)}	C = {S, A, E, F, H, L, M, I, J}
F = {(G, f=9+0=0), (D, f=9+2=11), (C, f=8+3=11), (O, f=9+1=10), (B, f=8+4=12), (N, f=8+2=10), (P, f=5+6=11)}	C = {S, A, E, F, H, L, M, I, J, K}
G - Remove Goal!	C = {S, A, E, F, H, L, M, I, J, K, G}
F = {(D, f=9+2=11), (C, f=8+3=11), (O, f=9+1=10), (B, f=8+4=12), (N, f=8+2=10), (P, f=5+6=11)}	

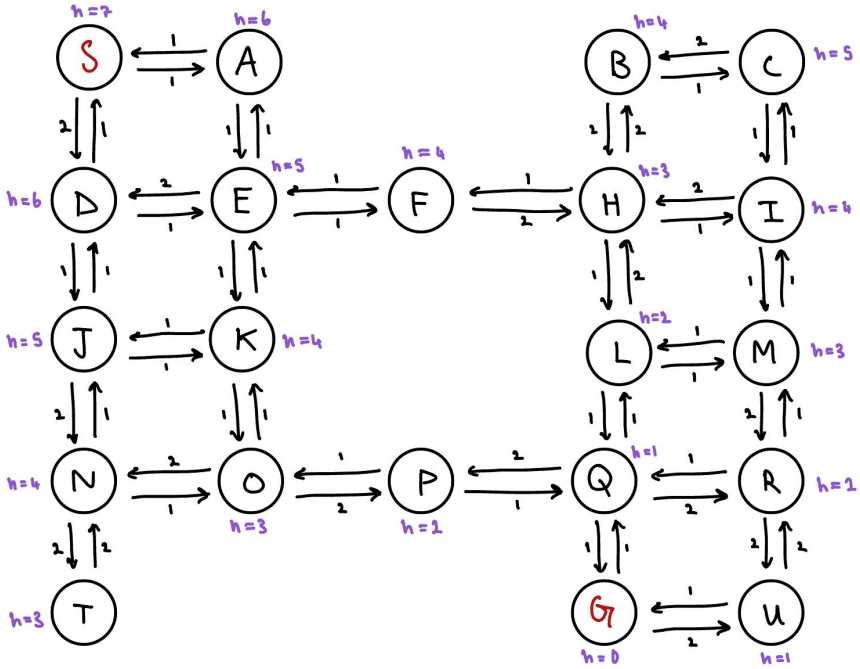
∴ States Expanded = {S, A, E, F, H, L, M, I, J, K}

∴ Goal Path = S, A, F, H, I, J, K, G

∴ Total Weight = 9

2nd Grid Environment:

Grid 2 represented as a graph:



Fringe list:	Closed Node List:
F = {(S, f =0+7= 7)}	C = { ∅ }
F = {(A, f =1+6= 7), (D, f =2+6= 8)}	C = {S}
F = {(E, f =2+5= 7), (D, f =2+6= 8)}	C = {S, A}
F = {(F, f =3+5= 8), (K, f =3+4= 7), (D, f =2+6= 8)}	C = {S, A, E}
F = {(H, f =4+4= 8), (K, f =3+4= 7), (D, f =2+6= 8)}	C = {S, A, E, F}
F = {(J, f =4+5=9), (O, f =4+3=7), (H, f =4+4= 8), (D, f =2+6= 8)}	C = {S, A, E, F, K}
F = {(N, f = 6+4=10), (P, f =6+2=8), (J, f =4+5=9), (H, f =4+4= 8), (D, f =2+6= 8)}	C = {S, A, E, F, K, O}
F = {(J, f =3+5= 8), (N, f = 6+4=10), (P, f =6+2=8), (J, f =4+5=9), (H, f =4+4= 8)}	C = {S, A, E, F, K, O, D}
F = {(B, f =7+4= 11), (I, f =6+4= 10), (L, f =6+2= 8), (J, f =3+5= 8), (N, f = 6+4= 10), (P, f =6+2= 8), (J, f =4+5= 9)}	C = {S, A, E, F, K, O, D, H}
F = {(N, f = 6+4= 10), (B, f =7+4= 11), (I, f =6+4= 10), (L, f =6+2= 8), (N, f = 6+4=10), (P, f =6+2= 8), (J, f =4+5= 9)}	C = {S, A, E, F, K, O, D, H, J}
F = {(M, f =7+3= 10), (Q, f =7+1= 8), (N, f = 6+4= 10), (B, f =7+4= 11), (I, f =6+4= 10), (N, f = 6+4= 10), (P, f =6+2= 8), (J, f =4+5= 9)}	C = {S, A, E, F, K, O, D, H, J, L}
F = {(Q, f =7+1= 8), (M, f =7+3= 10), (Q, f =7+1= 8), (N, f = 6+4= 10), (B, f =7+4= 11), (I, f =6+4= 10), (N, f = 6+4= 10), (J, f =4+5= 9)}	C = {S, A, E, F, K, O, D, H, J, L, P}
F = {(G, f =8+0=8), (R, f =9+2=11), (M, f =7+3= 10), (Q, f =7+1= 8), (N, f = 6+4= 10), (B, f =7+4= 11), (I, f =6+4= 10), (N, f = 6+4= 10), (J, f =4+5= 9)}	C = {S, A, E, F, K, O, D, H, J, L, P, Q}
G - Remove Goal!	C = {S, A, E, F, K, O, D, H, J, L, P, Q, G}
F = {(R, f =9+2=11), (M, f =7+3= 10), (Q, f =7+1= 8), (N, f = 6+4= 10), (B, f =7+4= 11), (I, f =6+4= 10), (N, f = 6+4= 10), (J, f =4+5= 9)}	

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

$\therefore$ Goal Path = S, A, E, F, H, L, Q, G

$\therefore$ Total Weight = 8

Part 1B (15 marks): Java Implementation of A* Search

Output:

- The console output from running the Main class should show the search process, including the current node being expanded, the goal node if found, the cost, the number of nodes expanded, and the path to the root.

Justification:

- GridState: This class is designed to represent the state in a grid-based search problem, with methods to check if the state is the goal and to calculate the heuristic value using the Manhattan distance.
- AStarSearch: Implements the A* search algorithm by defining how nodes are added to the frontier based on their heuristic values, ensuring the most promising nodes are expanded first.
- SearchProblem: Manages the search process, expanding nodes and checking for goal states, while keeping track of visited states to avoid cycles.
- SearchOrder: An interface that allows for different search strategies to be implemented and used interchangeably.
- FringeNode: Represents nodes on the frontier, including their cost and heuristic values, which are crucial for the A* search algorithm.
- IntState: A simple state representation using integers, demonstrating the flexibility of the search framework.
- Main: Demonstrates the setup and execution of the search problem, providing a clear example of how to use the implemented classes.

```
public class GridState implements State {
    //public char value;
    private final int x;
    private final int y;
    private final int goalX;
    private final int goalY;
    private final boolean goal;
    public GridState(int x, int y, int goalX, int goalY, boolean goal) {
        this.x = x;
        this.y = y;
        this.goalX = goalX;
        this.goalY = goalY;
        this.goal = goal;
    }
    public boolean isGoal() {
        return this.goal;
    }
    public int getHeuristic() {
        // System.out.println(this.x);
        // System.out.println(this.y);
        // System.out.println();
        return Math.abs(this.x - goalX) + Math.abs(this.y - goalY);
    }
    public String toString() {
        return "IntegerState [x=" + x + ", y=" + y + "]";
    }
}
public class AStarSearch implements SearchOrder {
    public void addToFringe(List<FringeNode> frontier, FringeNode parent, Set<ChildWithCost>
children) {
        for (ChildWithCost child : children) {
            // Create a new FringeNode for the child
            FringeNode newNode = new FringeNode(child.node, parent, child.cost);
            int i = 0;
            while (i < frontier.size() && frontier.get(i).getFValue() <= newNode.getFValue()) {
                i++;
            }
            frontier.add(i, newNode);
        }
    }
}
```

- Our Implementation works by creating a new node for each child, then using an insertion sort algorithm to sort and insert the fringe according to the F value of the current child node. The final result is a sorted fringe, in which the shortest path from the Start state to the Goal State can be determined.

```
public class Main {
    public static void main(String[] args) {
        // GRID 2 STATES:
        Node root = new Node(new GridState(0, 0, 4, 3, false));
        Node goal = new Node(new GridState(4, 3, 4, 3, true));
        Node nodeA = new Node(new GridState(0, 1, 4, 3, false));
        Node nodeB = new Node(new GridState(0, 3, 4, 3, false));
        Node nodeC = new Node(new GridState(0, 4, 4, 3, false));
        Node nodeD = new Node(new GridState(1, 0, 4, 3, false));
        Node nodeE = new Node(new GridState(1, 1, 4, 3, false));
        Node nodeF = new Node(new GridState(1, 2, 4, 3, false));
        Node nodeH = new Node(new GridState(1, 3, 4, 3, false));
        Node nodeI = new Node(new GridState(1, 4, 4, 3, false));
        Node nodeJ = new Node(new GridState(2, 0, 4, 3, false));
        Node nodeK = new Node(new GridState(2, 1, 4, 3, false));
        Node nodeL = new Node(new GridState(2, 3, 4, 3, false));
        Node nodeM = new Node(new GridState(2, 4, 4, 3, false));
        Node nodeN = new Node(new GridState(3, 0, 4, 3, false));
        Node nodeO = new Node(new GridState(3, 1, 4, 3, false));
        Node nodeP = new Node(new GridState(3, 2, 4, 3, false));
        Node nodeQ = new Node(new GridState(3, 3, 4, 3, false));
        Node nodeR = new Node(new GridState(3, 4, 4, 3, false));
        Node nodeT = new Node(new GridState(4, 0, 4, 3, false));
        Node nodeU = new Node(new GridState(4, 4, 4, 3, false));
        root.addChild(nodeA, 1);
        root.addChild(nodeD, 2);
        nodeA.addChild(nodeE, 1);
        nodeD.addChild(nodeE, 1);
        nodeD.addChild(nodeJ, 1);
        nodeE.addChild(nodeF, 1);
        nodeE.addChild(nodeK, 1);
        nodeJ.addChild(nodeK, 1);
        nodeJ.addChild(nodeN, 2);
        nodeN.addChild(nodeT, 2);
        nodeN.addChild(nodeO, 1);
        nodeK.addChild(nodeO, 1);
        nodeO.addChild(nodeP, 2);
        nodeP.addChild(nodeQ, 1);
        nodeF.addChild(nodeH, 2);
        nodeH.addChild(nodeB, 2);
        nodeH.addChild(nodeI, 1);
        nodeH.addChild(nodeL, 1);
        nodeB.addChild(nodeC, 1);
        nodeC.addChild(nodeI, 1);
        nodeI.addChild(nodeM, 1);
        nodeL.addChild(nodeM, 1);
        nodeL.addChild(nodeQ, 1);
        nodeM.addChild(nodeR, 2);
        nodeQ.addChild(nodeR, 2);
        nodeQ.addChild(goal, 1);
        nodeR.addChild(nodeU, 2);
        nodeU.addChild(goal, 1);
        // Run the search
        SearchOrder order = new AStarSearch();
        SearchProblem problem = new SearchProblem(order);
        problem.doSearch(root);
    }
}
```

- The Main Class holds the Node definitions and X, Y coordinates for each node in Grid1 and Grid2, above is Grid2. Grid1 has the same structure however is commented out. When running Main.java you will comment and uncomment the grids to run A* Search for both of them.
    - ❖ We checked if a given node is a goal by checking if the current X coordinate, current Y coordinate are the same as the goal X, Y coordinate.

Cartesian Graph representing the coordinates of each Node in Grid1, used to trace goal path.

0,0 S	0,1 A	0,2	0,3 B	0,4 C	0,5 D
1,0 E	1,1 F	1,2 H	1,3 I	1,4 J	1,5 K
2,0 L	2,1 M	2,2	2,3 N	2,4 O	2,5 G
3,0 P	3,1	3,2	3,3 Q	3,4 R	3,5 T

Cartesian Graph representing the coordinates of each Node in Grid2, used to trace goal path.

0,0 S	0,1 A	0,2	0,3 B	0,4 C
1,0 D	1,1 E	1,2 F	1,3 H	1,4 I
2,0 J	2,1 K	2,2	2,3 L	2,4 M
3,0 N	3,1 O	3,2 P	3,3 Q	3,4 R
4,0 T	4,1	4,2	4,3 G	4,4 U

Grid1 Output:

Goal Path found: **S A F H I J K G**
Total Weight: 9

```
Cost: 9
Nodes expanded: 10
Path to root:
- node: IntegerState [x=2, y=5]
- node: IntegerState [x=1, y=5]
- node: IntegerState [x=1, y=4]
- node: IntegerState [x=1, y=3]
- node: IntegerState [x=1, y=2]
- node: IntegerState [x=1, y=1]
- node: IntegerState [x=0, y=1]
- node: IntegerState [x=0, y=0]
```

Grid2 Output:

Goal Path found: **S A E K O P Q G**
Total Weight: 8

```
Cost: 8
Nodes expanded: 12
Path to root:
- node: IntegerState [x=4, y=3]
- node: IntegerState [x=3, y=3]
- node: IntegerState [x=3, y=2]
- node: IntegerState [x=3, y=1]
- node: IntegerState [x=2, y=1]
- node: IntegerState [x=1, y=1]
- node: IntegerState [x=0, y=1]
- node: IntegerState [x=0, y=0]
```

Part 2: Automated Planning

Part 2A: Modelling the Domain

The **domain.pddl** file defines the basic domain for the underwater robotic inspection mission. Here's a detailed breakdown of its structure:

Types:

- uuv: Represents the Unmanned Underwater Vehicle.
- ship: Represents the ship carrying the UUV.
- location: Represents various locations in the underwater environment.
- data: Represents data collected by the UUV.
- sample: Represents samples collected by the UUV.

Predicates:

- (at ?x - (either uuv ship sample) ?l - location): Indicates that an object (UUV, ship, or sample) is at a specific location.
- (connected ?l1 - location ?l2 - location): Indicates that two locations are connected.
- (has-image ?u - uuv ?l - location): Indicates that the UUV has captured an image at a specific location.
- (has-sonar ?u - uuv ?l - location): Indicates that the UUV has conducted a sonar scan at a specific location.
- (has-sample ?u - uuv): Indicates that the UUV has collected a sample.
- (image-transmitted ?u - uuv ?s - ship ?l - location): Indicates that the UUV has transmitted image data to the ship from a specific location.
- (sonar-transmitted ?u - uuv ?s - ship ?l - location): Indicates that the UUV has transmitted sonar data to the ship from a specific location.
- (sample-stored ?u - uuv ?s - ship ?sm - sample): Indicates that a sample has been stored on the ship.
- (deployed ?u - uuv): Indicates that the UUV has been deployed.

The **domain1.pddl** file extends the basic domain to include additional elements such as engineers and specific ship locations. Here's a detailed breakdown of its structure:

Types:

- Including all previous types from domain.pddl
- engineer: Represents engineers who assist in deploying and managing the UUVs.
- ship-location: Represents specific locations on the ship, such as the bay and control centre.

Constants:

- uuv1, uuv2: Instances of the UUV.
- engineer1, engineer2: Instances of engineers.
- bay1, bay2, control-centre1, control-centre2: Specific locations on the ship.

Predicates:

- Including all previous predicates from domain.pddl
- (engineer-at ?e - engineer ?sl - ship-location): Indicates that an engineer is at a specific location on the ship.
- (at-ship-location ?sl - ship-location ?s - ship): Indicates that a specific location is part of a ship.

Actions:

- Deploy UUV: Deploys the UUV from the ship to a specific location, requiring the presence of an engineer at the bay.
- Move UUV: Moves the UUV from one location to another.
- Capture Image: Captures an image at a specific location.
- Conduct Sonar Scan: Conducts a sonar scan at a specific location.
- Collect Sample: Collects a sample from a specific location.
- Transmit Image Data: UUV Transmits image data to the ship, given the engineer is at the control centre.
- Transmit Sonar Data: UUV Transmits sonar data to the ship, given the engineer is at the control centre.
- Store Sample: Stores a collected sample on the ship, given the engineer is at the bay.
- Move Engineer: Moves an engineer between locations on the ship.

Location of the Control Centre:

In the domain1.pddl file, the control centre is a specific location on the ship where engineers must be present to transmit data from the UUV to the ship. The control centre locations are defined as follows:

- Control Centre 1 (control-centre1): This is a specific location on ship1.
- Control Centre 2 (control-centre2): This is a specific location on ship2.

These control centres are crucial for the operations involving data transmission.

Dual-BFWS-FFparser:

We Utilised The FF (Fast-Forward) planner which is a heuristic forward search algorithm that efficiently generates plans by using a heuristic estimate derived from a plan graph. This heuristic guides the search towards the goal state, reducing the number of nodes expanded and improving performance. The FF planner is known for its speed and effectiveness, making it a suitable choice for complex planning problems like underwater robotic inspection missions, where rapid and reliable plan generation is crucial.

Part 2B: Modelling the Problems

Task 2.1: Problem 1

Mission Goals:

- Save an Image at waypoint 3
- Save a Sonar at waypoint 4

```
Plan found:
0.00000: (DEPLOY-UUV UUV1 SHIP1 WP3)
0.00100: (CAPTURE-IMAGE UUV1 WP3)
0.00200: (TRANSMIT-IMAGE-DATA UUV1 SHIP1 WP3)
0.00300: (MOVE-UUV UUV1 WP3 WP4)
0.00400: (CONDUCT-SONAR-SCAN UUV1 WP4)
0.00500: (TRANSMIT-SONAR-DATA UUV1 SHIP1 WP4)
Metric: 0.005
Makespan: 0.005
States evaluated: undefined
Planner found 1 plan(s) in 3.746secs.
```

Task 2.2: Problem 2

Mission Goals:

- Save an Image at waypoint 5
- Save a Sonar at waypoint 3
- Collect a sample from waypoint 1

```
Plan found:
0.00000: (DEPLOY-UUV UUV1 SHIP1 WP1)
0.00100: (COLLECT-SAMPLE UUV1 WP1 SAMPLE1)
0.00200: (STORE-SAMPLE UUV1 SHIP1 SAMPLE1 WP1)
0.00300: (MOVE-UUV UUV1 WP1 WP4)
0.00400: (MOVE-UUV UUV1 WP4 WP3)
0.00500: (CONDUCT-SONAR-SCAN UUV1 WP3)
0.00600: (TRANSMIT-SONAR-DATA UUV1 SHIP1 WP3)
0.00700: (MOVE-UUV UUV1 WP3 WP5)
0.00800: (CAPTURE-IMAGE UUV1 WP5)
0.00900: (TRANSMIT-IMAGE-DATA UUV1 SHIP1 WP5)
Metric: 0.009000000000000001
Makespan: 0.009000000000000001
States evaluated: undefined
Planner found 1 plan(s) in 3.802secs.
```

Task 2.3: Problem 3

Mission Goals:

- Save an Image at waypoint 3 and waypoint 2
- Save a Sonar at waypoint 4 and waypoint 6
- Collect a sample from waypoint 1 and waypoint 5

```
Plan found:
0.00000: (CAPTURE-IMAGE UUV1 WP2)
0.00100: (TRANSMIT-IMAGE-DATA UUV1 SHIP1 WP2)
0.00200: (DEPLOY-UUV UUV2 SHIP2 WP5)
0.00300: (COLLECT-SAMPLE UUV2 WP5 SAMPLE2)
0.00400: (STORE-SAMPLE UUV2 SHIP2 SAMPLE2 WP5)
0.00500: (MOVE-UUV UUV1 WP2 WP3)
0.00600: (CAPTURE-IMAGE UUV1 WP3)
0.00700: (TRANSMIT-IMAGE-DATA UUV1 SHIP1 WP3)
0.00800: (MOVE-UUV UUV2 WP5 WP6)
0.00900: (CONDUCT-SONAR-SCAN UUV2 WP6)
0.01000: (TRANSMIT-SONAR-DATA UUV2 SHIP2 WP6)
0.01100: (MOVE-UUV UUV2 WP6 WP4)
0.01200: (CONDUCT-SONAR-SCAN UUV2 WP4)
0.01300: (TRANSMIT-SONAR-DATA UUV2 SHIP2 WP4)
0.01400: (MOVE-UUV UUV1 WP3 WP5)
0.01500: (MOVE-UUV UUV1 WP5 WP6)
0.01600: (MOVE-UUV UUV1 WP6 WP4)
0.01700: (MOVE-UUV UUV1 WP4 WP2)
0.01800: (MOVE-UUV UUV1 WP2 WP1)
0.01900: (COLLECT-SAMPLE UUV1 WP1 SAMPLE1)
0.02000: (MOVE-UUV UUV1 WP1 WP2)
0.02100: (STORE-SAMPLE UUV1 SHIP1 SAMPLE1 WP2)
Metric: 0.02100000000000001
Makespan: 0.02100000000000001
States evaluated: undefined
Planner found 1 plan(s) in 3.597secs.
```

Part 2C: Extension

Task 3.1: Problem 4

Mission Goals:

- Add locations on the ship: the bay and the control centre. The engineer can walk between these two locations. Uuv's can only deploy/store sample from bay and transmit to control-centre.

```
Plan found:
0.00000: (CAPTURE-IMAGE UUV1 WP2)
0.00100: (TRANSMIT-IMAGE-DATA UUV1 SHIP1 WP2 ENGINEER1 CONTROL-CENTRE1)
0.00200: (MOVE-UUV UUV1 WP2 WP3)
0.00300: (CAPTURE-IMAGE UUV1 WP3)
0.00400: (TRANSMIT-IMAGE-DATA UUV1 SHIP1 WP3 ENGINEER1 CONTROL-CENTRE1)
0.00500: (DEPLOY-UUV UUV2 SHIP2 WP5 ENGINEER2 BAY2)
0.00600: (COLLECT-SAMPLE UUV2 WP5 SAMPLE2)
0.00700: (STORE-SAMPLE UUV2 SHIP2 SAMPLE2 WP5 ENGINEER2 BAY2)
0.00800: (MOVE-ENGINEER ENGINEER1 CONTROL-CENTRE1 BAY1 SHIP1)
0.00900: (MOVE-UUV UUV2 WP5 WP6)
0.01000: (MOVE-ENGINEER ENGINEER2 BAY2 CONTROL-CENTRE2 SHIP2)
0.01100: (CONDUCT-SONAR-SCAN UUV2 WP6)
0.01200: (TRANSMIT-SONAR-DATA UUV2 SHIP2 WP6 ENGINEER2 CONTROL-CENTRE2)
0.01300: (MOVE-UUV UUV2 WP6 WP4)
0.01400: (CONDUCT-SONAR-SCAN UUV2 WP4)
0.01500: (TRANSMIT-SONAR-DATA UUV2 SHIP2 WP4 ENGINEER2 CONTROL-CENTRE2)
0.01600: (MOVE-UUV UUV1 WP3 WP5)
0.01700: (MOVE-UUV UUV1 WP5 WP6)
0.01800: (MOVE-UUV UUV1 WP6 WP4)
0.01900: (MOVE-UUV UUV1 WP4 WP2)
0.02000: (MOVE-UUV UUV1 WP2 WP1)
0.02100: (COLLECT-SAMPLE UUV1 WP1 SAMPLE1)
0.02200: (MOVE-UUV UUV1 WP1 WP2)
0.02300: (STORE-SAMPLE UUV1 SHIP1 SAMPLE1 WP2 ENGINEER1 BAY1)
Metric: 0.023000000000000013
Makespan: 0.023000000000000013
States evaluated: undefined
Planner found 1 plan(s) in 3.54secs.
```

Video Demo Link:

https://drive.google.com/file/d/1OGDSnxIrgXKxswBHC4cipKiQOqLF2c-h/view?usp=sharing