# **Design Document - 2022 S1 ACI A1**

The PSA (Problem, Solution, and Assessment) design environment of finding the best path to a heart using the **A\*** algorithm can be defined as follows:

## Problem Statement:

There are two agents named **R1** and **G1**. Both are searching for a "**heart**" as shown in the below configuration as “**H**” that gives everlasting power. Both agents are trying to reach the heart. In this process many obstacles may be encountered to reach the heart. Help them in finding the best path to reach the heart from any arbitrary start positions. [Dynamically fetch the start position while executing the code]

Given a 2D plane or grid representing the search space, find the optimal path to a heart goal while minimizing the sum of the costs of the actions taken using Manhattan distance, penalties, and the heuristic estimate of the distance from each node to the heart.

## Solution:

The document aims to develop an optimal solution to the problem to find the best cost path to the goal node using A-star algorithm.

## Overview:

Multi agent search to identify best path from any arbitrary point using ‘**A\***’ algorithm.

Many real-life problems of scientific importance involve finding a path with the minimum cost between two nodes – a source and destination in any space or area E.g., Google maps, routing of network traffic, navigation through a maze, etc. Dijkstra Algorithm is used in Google maps to find shortest path between two nodes. In our case study we would be using ‘**A\***’ algorithm which can be seen as an extended version of the Dijkstra algorithm.

This criterion is linked to a Learning Outcome Identification & Design of the environment (PEAS)

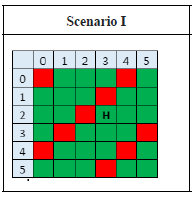
## PEAS Environment:

The PEAS (*Performance measure, Environment, Actuators, Sensors*) framework can be applied to finding the best path to a heart using the **A\*** algorithm as follows:

|  |  |
| --- | --- |
| Performance measure | Least amount of cost (and with lesser visits if possible). The optimal path to the heart, minimizing the sum of the costs of the actions taken and the heuristic estimate of the distance from the node to the heart.  In this case, color penalty points (+10 / -10) |
| Environment | The search space represented as a 2D plane or grid with Green rooms and Red rooms.  The search space is fully observable, deterministic, static & discrete. |
| Actuators | They perform the role of controlling and moving a system.  Display of marking agents visit during the actions that can be taken to move from one node to another, such as up, down, left, right, or diagonal moves. |
| Sensors | Green and Red colored rooms. The cost of each action, the heuristic estimate of the distance from each node to the heart, and the state of each node (open, closed, start, end). |

## Search Environment

The search environment consists of two 6x6 grids that has green and red colored nodes (rooms). The placement of the colored nodes vary for both the scenarios.



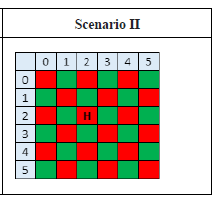
Scenario 1:

In Scenario 1, the red nodes are located as given in the diagram below.

The goal node for this scenario is (2, 3) placed in a green room.

Scenario II

Scenario II is a checker board with red and green rooms placed adjacent to each other.



The goal node for this scenario is (2,2) placed in a green room.

In this scenario, a node is surrounded by different colored rooms in all directions. Moving to the goal node from any node will require moving through nodes of both the colors.

## Step cost and Heuristic cost:

The algorithm implements **A\*** search algorithm to reach the goal node, using the Manhattan distance and color penalty as the heuristic cost, and base cost (1) and obstacle penalty as the step cost for determining the next room for movement. The agent will move to the room that provides the lowest cost estimation using the evaluation function given below:

The evaluation function for **A\*** algorithm is given by

***f(n) = g(n) + h(n)***

Where:

* ***g(n)*** is the cost from the start node to the current node **n**.
* ***h(n)*** is the estimated cost from the current node **n** to the **goal node**.

The **A\*** algorithm uses this evaluation function to determine the next node to expand, choosing the node with the lowest **f(n)** value. The **h(n)** value is the heuristic function that provides an estimate of the distance between the current node and the goal node.

|  |
| --- |
| Evaluation function **f(n) = g(n) + h(n)** |
| **g(n)** - the cost to reach the node |
| **h(n)** - the optimistic expected cost to go from node to goal |
| **f(n)** - estimated cost of cheapest path through node **n** |

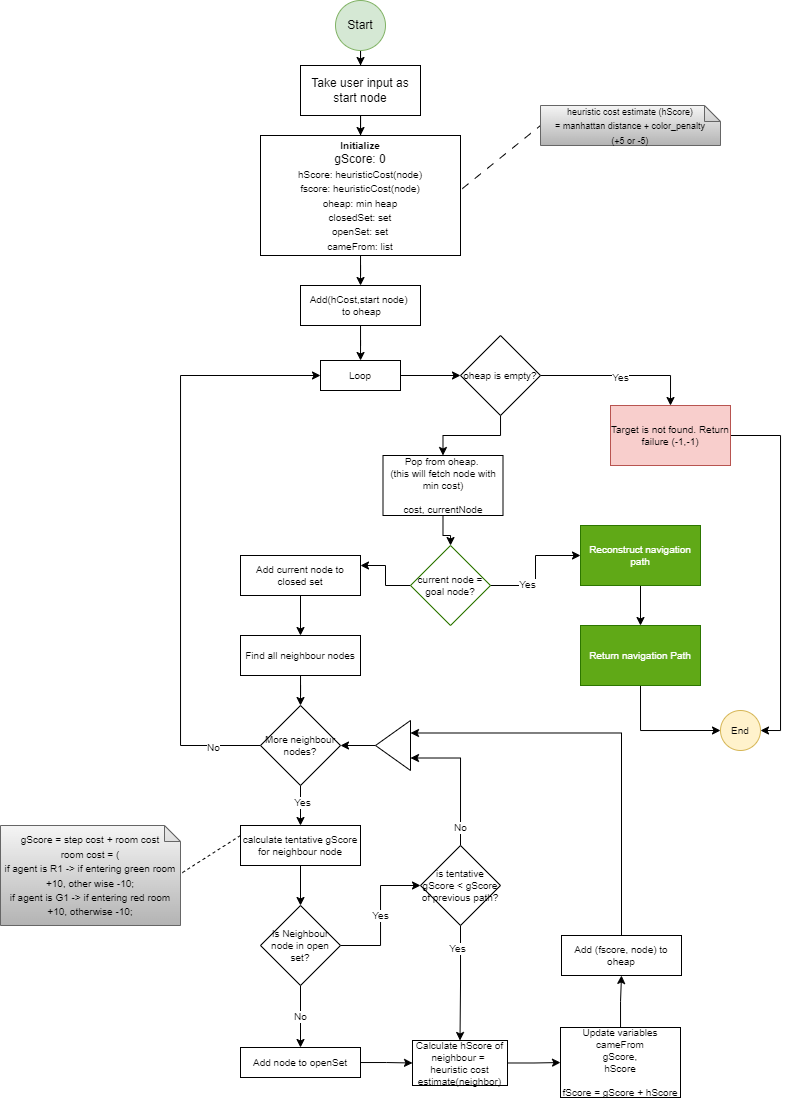
**Given:**

|  |  |
| --- | --- |
| **Items** | **Description** |
| Agents | R1, G1  Both these agents are independent such that they are neither competing nor collaborating with each other.  Both the agents may take different route depending on the cost evaluation function taking into account the penalties and the dynamic starting point. |
| Goals | Heart in **Green** for Scenario 1 – (2,3)  Heart in **Red** for Scenario 2 – (2,2) |
| Searching Algorithm | A\* Search Algorithm |
| Penalty | |
| Color penalty | If color of the goal node and the node under consideration is same, then **+5**  If goal node and node under consideration are having different colors, then **-5** |
| R1 obstacle | For moving to Green room, penalty = **+10**,  For moving to Red room, penalty = **-10** |
| G2 obstacle | For moving to Red room, penalty = +10  For moving to Green room, penalty = -10 |
| Heuristic Cost | Manhattan distance + Color penalty |

## Search Algorithm

1. Initialize a min heap with the starting node and its cost (g(n) + h(n), where g(n) is the cost from the start node to n and h(n) is the heuristic estimate of the cost from n to the goal node). For the start node g(n) will evaluate to 0 as no step cost is involved.
2. Repeat the following steps until the min heap is empty or the goal node is found:
   1. Remove the node with the lowest priority (g(n) + h(n)) from the priority queue.
   2. If this node is the goal node, return the path from the starting node to the goal node.
   3. Generate the children nodes of the current node by moving in the four directions (North, South, East, and West).
   4. For each child node, calculate its cost (g(n) + h(n)) and add it to the priority queue.
   5. Update the cost of the child node if it is already in the priority queue and the new cost is lower.
3. If the priority queue is empty and the goal node is not found, return failure.

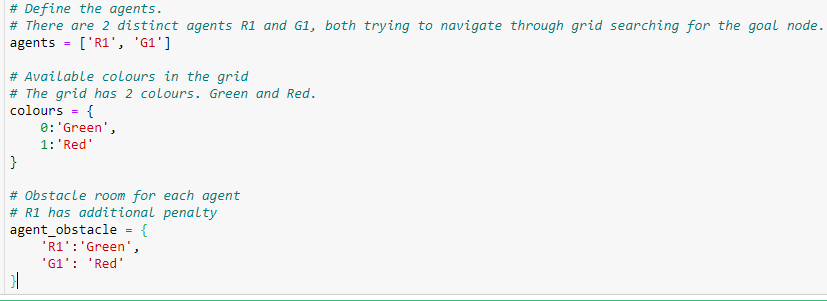
## Flow chart of the algorithm implementation



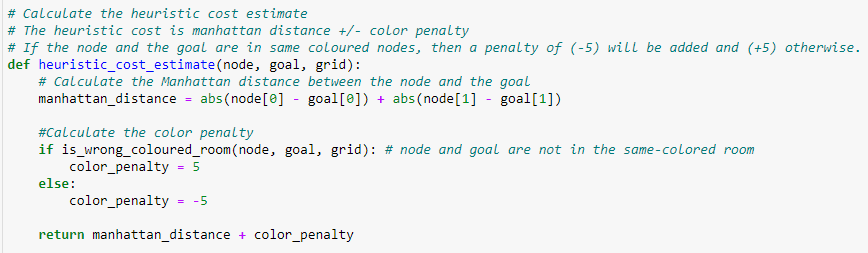
## Implementation

The solution is implemented in python using Jupyter Notebook environment.

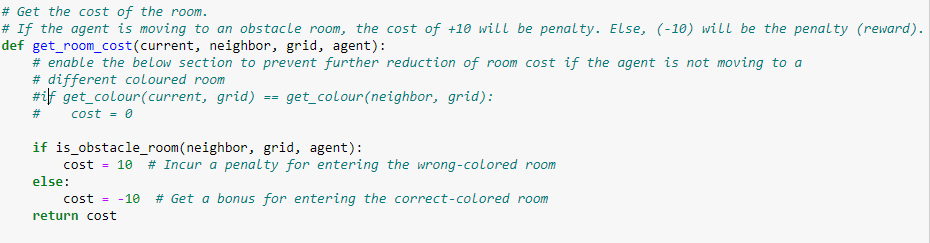
The solution initially defines the agents, obstacles of both the agents as maps.



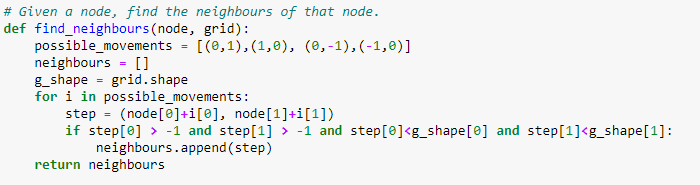
Heuristic function uses Manhattan distance and the room color penalty to determine the heuristic cost estimate.



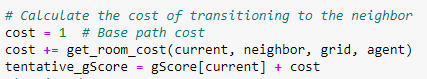
The room cost is calculated by considering the color of the room to which the agent is moving and applying the appropriate color penalty as given above.



The list of possible neighbors is evaluated by determining the possible movements of the agent. Only 4 movements are allowed for the agent – North, South, East and West. Movements are restricted only by edge of the grid. Otherwise agent can move to any of the available neighboring nodes.



The tentative gCost is calculated by considering the base path cost of 1 and the room cost.



The evaluation cost is calculated by taking into account the heuristic cost (hcost) and the gScore.

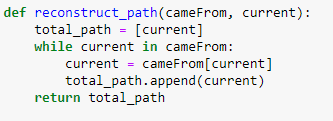


Each valid node is added to the min heap using the fscore value as the sorting value.



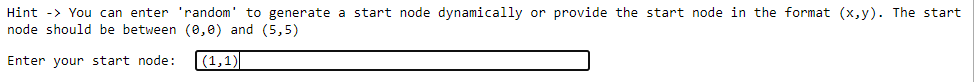
Each opened node is iterated till the goal node is reached. If the goal node could not be reached even after min heap is empty, ‘failure’ result is retuned to indicate that goal could not be identified.

When the goal node is identified, the best path (having lowest cost) to reach the node is unraveled by moving through the list that stores the path followed by the agent.



## User Input for start node

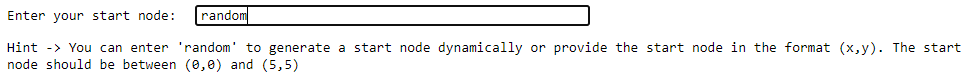
The program allows user to provide the start node as any arbitrary node in the grid. Start point need to be specified as a valid tuple.



The start point given will be passed to both the agents and both scenarios to trace the best path.



Alternatively, the user can specify value ‘random’ let the program generate a random start point.



The program generates a random start point within the boundaries of the grid when input ‘random’ is provided.



### Input validations

Input validation is pivotal for the successful execution of the program. Hence the below validations are implemented to ensure that invalid values are not provided to the search algorithm.

* 1. The input value should be tuple or ‘random’
  2. The tuple values should be numerical and within the boundaries of the grid.

## Evaluation

For the purpose of testing and evaluating the algorithm, node with coordinates **(1, 1)** is considered as the starting point.

**Agent G1:**

**Grid 1 (Scenario 1)**

Starting node: (1, 1)

Goal Node: (2, 3)

Final Path Taken by G1 through Grid 1: **(1, 1)->(1, 2)->(0, 2)->(0, 3)->(1, 3)->(2, 3)**



---navigation logs—

| **Node** | **Step Cost** | **Heuristic Cost** | **Total Cost** | **Node Color** | **Goal Node?** |
| --- | --- | --- | --- | --- | --- |
| **0** | (1, 1) | 0 | -2 | -2 | Green | 0 |
| **1** | (1, 2) | -9 | -3 | -12 | Green | 0 |
| **2** | (0, 2) | -18 | -2 | -20 | Green | 0 |
| **3** | (0, 3) | -27 | -3 | -30 | Green | 0 |
| **4** | (0, 1) | -27 | -1 | -28 | Green | 0 |
| **5** | (2, 1) | -9 | -3 | -12 | Green | 0 |
| **6** | (2, 0) | -18 | -2 | -20 | Green | 0 |
| **7** | (1, 0) | -27 | -1 | -28 | Green | 0 |
| **8** | (3, 0) | -27 | -1 | -28 | Green | 0 |
| **9** | (0, 1) | -27 | -1 | -10 | Green | 0 |
| **10** | (1, 0) | -27 | -1 | -10 | Green | 0 |
| **11** | (1, 3) | -16 | 6 | -10 | Red | 0 |
| **12** | (2, 3) | -25 | -5 | -30 | Green | 1 |

**Agent R1**

**Grid 1 (Scenario 1)**

Starting node: (1, 1)

Goal Node: (2, 3)

Final Path Taken by R1 through Grid 1: **(1, 1)->(1, 2)->(1, 3)->(2, 3)**

---navigation logs—

|  | **Node** | **Step Cost** | **Heuristic Cost** | **Total Cost** | **Node Colour** | **Goal Node?** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | (1, 1) | 0 | -2 | -2 | Green | 0 |
| **1** | (1, 2) | 11 | -3 | 8 | Green | 0 |
| **2** | (1, 3) | 2 | 6 | 8 | Red | 0 |
| **3** | (2, 1) | 11 | -3 | 8 | Green | 0 |
| **4** | (2, 2) | 2 | 6 | 8 | Red | 0 |
| **5** | (2, 3) | 13 | -5 | 8 | Green | 1 |

**Agent G1:**

**Scenario 2 (Grid 2)**

Starting node: (1, 1)

Goal Node: (2, 2)

Final Path Taken by G1 through Grid 2: **(1, 1)->(1, 2)->(2, 2)**

---navigation logs--

|  | **Node** | **Step Cost** | **Heuristic Cost** | **Total Cost** | **Node Colour** | **Goal Node?** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | (1, 1) | 0 | -3 | -3 | Red | 0 |
| **1** | (1, 2) | -9 | 6 | -3 | Green | 0 |
| **2** | (2, 1) | -9 | 6 | -3 | Green | 0 |
| **3** | (2, 2) | 2 | -5 | -3 | Red | 1 |

**Agent R1:**

**Scenario 2 (Grid 2)**

Starting node: (1, 1)

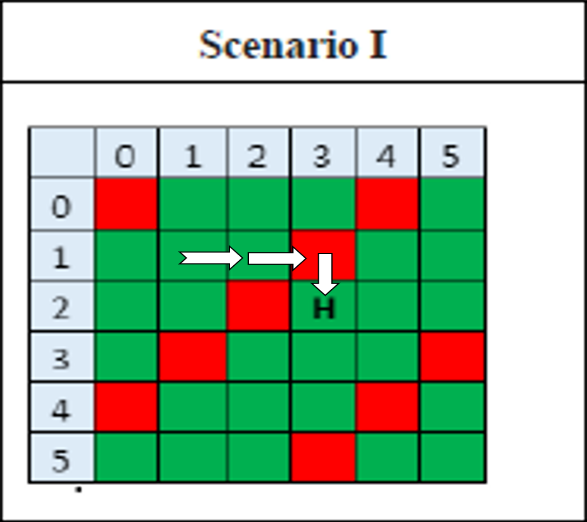
Goal Node: (2, 2)

Final Path Taken by R1 through Grid 2: (1, 1)->(1, 2)->(2, 2)

---navigation logs—

|  | **Node** | **Step Cost** | **Heuristic Cost** | **Total Cost** | **Node Colour** | **Goal Node?** |
| --- | --- | --- | --- | --- | --- | --- |
| **0** | (1, 1) | 0 | -3 | -3 | Red | 0 |
| **1** | (1, 2) | 11 | 6 | 17 | Green | 0 |
| **2** | (2, 2) | 2 | -5 | -3 | Red | 1 |

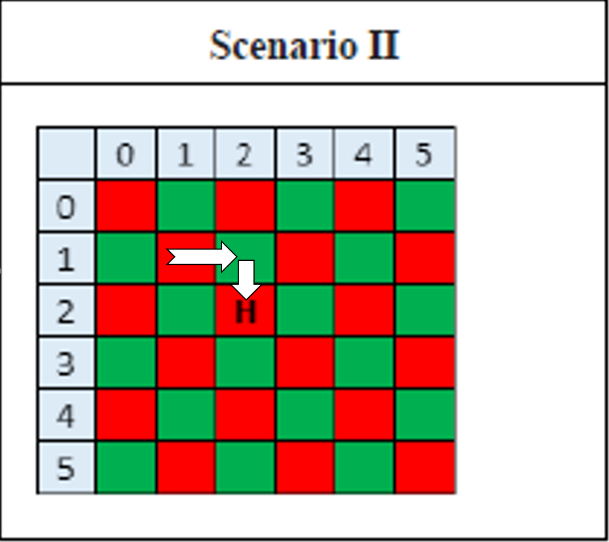
## Performance Assessment:

**Final path R1 through Scenario I**:

Starting node (1, 1) Goal Node (2, 3)

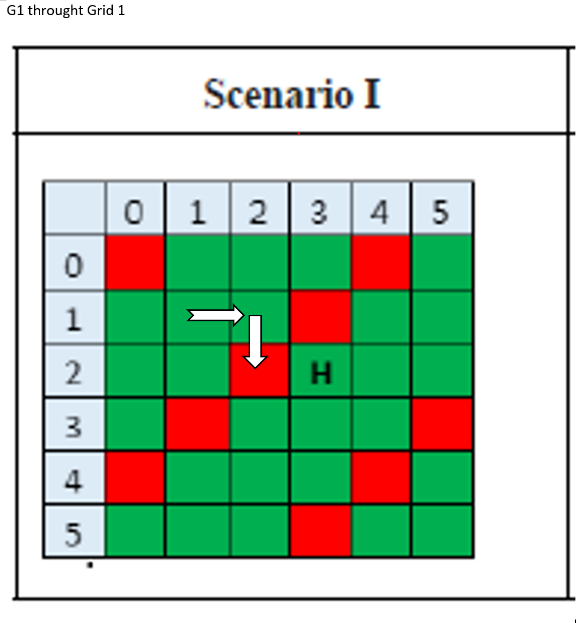
Final Path Taken by R1 through Grid 1 (1, 1) -> (1, 2) -> (1, 3) -> (2, 3)

**Final path G1 through Scenario II:**

Starting node (1, 1) to reach Goal Node (2, 2)

Final Path Taken by G1 through Grid 1 (1, 1) -> (1, 2) -> (2, 2)

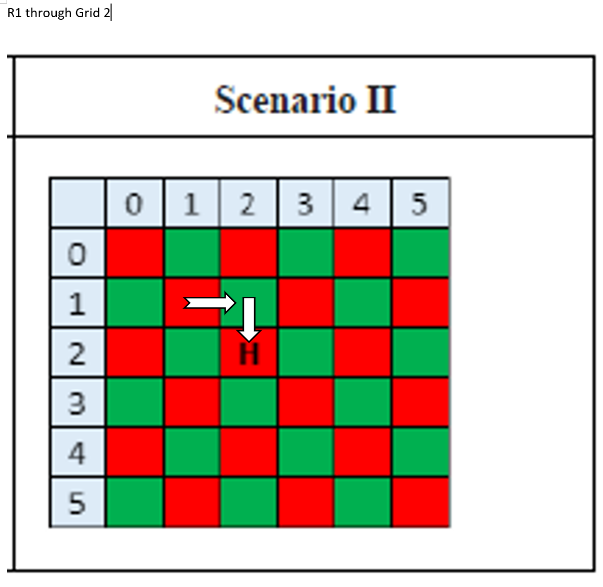
**Final path G1 through Scenario II:**

**** Starting node to reach Goal Node : (1, 1)

Goal Node: (2, 2)

Final Path Taken by G1 through Grid 2: **(1, 1)->(1, 2)->(2, 2)**

**Final path R1 through Scenario II:**



Starting node (1, 1) to reach Goal Node (2, 2)

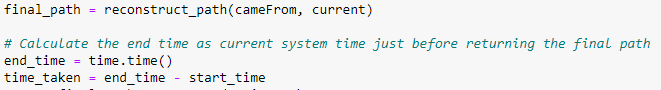
Final Path Taken by R1 through Grid 2 **(1, 1) -> (1, 2) -> (2, 2)**

### Space and time Complexity

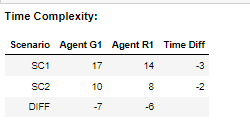
Time complexity is calculated by measuring time taken for execution of the algorithm for each of the scenarios given in the problem statement.



The final time is calculated when the goal node is identified or when the program returns a failure.



The below table summarizes the time complexity calculated for all the 4 execution scenarios



Space Complexity is calculated by considering the number of nodes opened for the search. A counter is maintained in the program, which is incremented for each node opened.







The total space is returned along with the final search path.

The space complexity observation of all the four scenarios is given below:

Graphical user interface, text, application

Description automatically generated

## Comparative analysis based on time & space study:

* Both the agents perform better in terms of time and space complexity in scenario 2 (grid 2).
* In scenario 1, the performance of the agents depends heavily on the start node of the agents. If G1 agent is placed above the red diagonal rooms in the center, more number of rooms are opened to reach the node. However if the agent G1 is placed below the red diagonal rooms, it reaches the goal quickly
* For both the scenarios, agent R1, gives search path in terms of both time and cost efficiency
* In scenario 2, the checker board pattern negates the room penalty and the color penalty as it moves across rooms. As moving across rooms are negating the room penalties, the Manhattan distance take prominence in determining the best path. This is evident from the evaluation results where we can see scenario 2 giving better time and cost efficiency.