

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DEPARTMENT OF COMPUTING AND INFORMATION SYSTEMS**

**SCHOOL OF ENGINEERING AND TECHNOLOGY**

|  |  |  |
| --- | --- | --- |
| **COURSE** | **:** | **ARTIFICIAL INTELLIGENCE** |
| **COURSE CODE** | **:** | **CSC3206** |
| **ACADEMIC SESSION** | **:** | **April 2024** |
| **DEADLINE** | **:** | **19th July 2024 by 23:59** |

**Assignment 2 – Group 8**

|  |  |  |
| --- | --- | --- |
|  | **NAME** | **STUDENT ID** |
| 1 | Tan Jun Rong | 21041967 |
| 2 | Yap Jay Ann | 21024765 |
| 3 | Yeong Meng Li | 21018429 |
| 4 | Lim Xiwei | 21045596 |

**Lecturer’s Remark** (Use additional sheet if required)

List down the names and the student ID here.

I **Tan Jun Rong, Yap Jay Ann, Yeong Meng Li, Lim Xiwei** (Student’s Name) **21041967, 21024765, 21018429, 21045596** (Student ID) received the assignment and read the comments.

Rong **17/06/2024**, A black background with a black square

Description automatically generated with medium confidence **17/06/2024**, mengli **17/06/2024**,  **Shape

Description automatically generated with low confidence****17/06/2024** (Signature/Date)

**Academic Honesty Acknowledgement**

“I **Tan Jun Rong, Yap Jay Ann, Yeong Meng Li, Lim Xiwei** (Student’s Name) verify that this paper contains entirely my own work. I have not consulted with any outside person or materials other than what was specified (an interviewee, for example) in the assignment or the syllabus requirements. Further, I have not copied or inadvertently copied ideas, sentences, or paragraphs from another student. I realize the penalties *(refer to page 16, 5.5, Appendix 2, page 44 of the student handbook diploma and undergraduate programme)* for any kind of copying or collaboration on any assignment.”

Rong **17/06/2024**, A black background with a black square

Description automatically generated with medium confidence **17/06/2024**, mengli **17/06/2024**,  **Shape

Description automatically generated with low confidence17/06/2024** (Student’s signature / Date)

Table of Contents

[**1.0** **Algorithm Implementation** 3](#_Toc171971204)

[1.1 State Representation and Assumptions 3](#_Toc171971205)

[1.2 Cost and Heuristic Function 3](#_Toc171971206)

[1.3 Transition Model 4](#_Toc171971207)

[1.4 Goal Searching 5](#_Toc171971208)

[**2.0** **Solution Representation** 6](#_Toc171971209)

[**3.0** **Algorithm Evaluation** 8](#_Toc171971210)

[3.1 Performance and Efficiency 8](#_Toc171971211)

[3.2 Limitations and Existing Issues 9](#_Toc171971212)

# **Algorithm Implementation**

# **State Representation and Assumptions**

The problem is formulated as a treasure hunt on a hexagonal grid, where the goal is to find the shortest path to collect all treasures while navigating through different cell types, such as traps and rewards. Each cell on the grid is uniquely identified using cube coordinates (x, y, z) (**Figure 1**). The hexagonal grid map is defined using a dictionary called *‘HEX\_GRID’*. The state space includes the player's current position on the grid and the types of cells encountered.

A hexagons with numbers and letters

Description automatically generated

**Figure 1.** Hex-Grid Coordination

The default energy cost for traversal is 1kJ per step and moving to an adjacent cell costs 1 step. Our assumptions for trap and reward effects are as follows: Reward 1 continuously halves the energy cost based on the previous step's energy expenditure and Reward 2 reduces the current step's energy cost by half and updates the default step cost variable. Trap 1 has the opposite effect of Reward 1, doubling the energy cost instead of halving it. Similarly, Trap 2 operates inversely to Reward 2. Trap 3 teleports the player two cells forward in their previous moving direction, and Trap 4 triggers a game-over scenario by removing all uncollected treasures from the map.

# **Cost and Heuristic Function**

A\* algorithm focuses on finding the lowest evaluation function, *f(n)*, which is calculated using the formula *f(n) = g(n) + h(n)*. Here, *g(n)* denotes the cumulative path cost from the starting node to the current node. This is implemented in the ‘*g*’ attribute in the Cell class. Each time a new cell is added to the path, the cost ‘*g*’ is incremented by the *‘step\_cost’* value to reflect the energy spent to move from one cell to another. The *‘step\_cost’* is part of the state object and can be modified by traps and rewards encountered along the path.

*h(n)* represents the heuristic function estimating the cost of the cheapest path from the current node to the goal node. To determine an admissible heuristic, we use the Manhattan distance denoted by the formula *h = Max(|x1 – x2|, |y1 – y2|, |z1 – z2|)*. This function computes the maximum absolute difference between corresponding coordinates (x, y, z) of two points, providing an estimation of distance suitable for a hexagonal grid in three-dimensional space. If the program is not searching for the nearest treasure, the heuristic is implemented with a strong preference for reward cells, ensuring they are prioritized by returning − ∞. In contrast, the heuristic strongly avoids trap cells by returning + ∞, preventing the algorithm from exploring them unless necessary. When the program searches for the nearest treasure, the heuristic function calculates values based on the Manhattan distance regardless of the cell type (**Figure 2**).

A screen shot of a computer code

Description automatically generated

**Figure 2.** Code snippet showing heuristic function

Using the evaluation function formula,

1. We determine which cell to move to by selecting the one with the lowest f value from the frontier, where f = g + h.
2. The node with the lowest f cost is removed from the frontier and expanded to become the next current node. This involves generating its neighbouring cells.
3. If a neighbouring node is not already in the frontier, not being explored, is within the hexagonal grid boundaries, and is not an obstacle, it is added to the frontier.
4. If a neighbouring node is already in the frontier but has a lower cost, its cost is updated with the new lower cost.
5. The algorithm continues to expand the nodes, update the costs, and move nodes between the frontier and the explored list. This cycle repeats for every node until the goal is reached.

# **Transition Model**

In our implementation of the A\* algorithm for the hex grid map, the transition model is a crucial component that defines how the player moves from one state to another. Each move expands from the current cell to its neighbouring cells, following the six possible directions in a hexagonal grid. Movement across the grid follows specific directional changes, including south-east, north-east, north, north-west, south-west, and south (**Figure 3**). Each direction is represented by coordinate adjustments defined in the *'DIRECTIONS'* list. Neighbouring cells are identified based on these directional coordinates relative to the current position.

A diagram of a diagram

Description automatically generated

**Figure 3.** Movable Directions

This design was tailored to handle the grid’s distinct characteristics, accommodating various constraints and effects imposed by traps and rewards scattered throughout the map. When encountering a trap, the transition model triggers specific effects, such as teleportation or adjustments to movement costs. Similarly, rewards modify the player’s state by reducing step or energy costs. The transition model also dynamically updates the goal to the nearest treasure, continuously refining the path based on immediate objectives. In addition, adjustments have been made to prioritize reward cells over treasure cells only when the reward cell is adjacent (**Figure 4**). This change is expected to significantly reduce the cost required to complete the game.

A screen shot of a computer code

Description automatically generated

**Figure 4.** Code snippet showing optimization changes

# **Goal Searching**

The goal searching for this implementation of A\* representation involves searching for multiple (4) treasures and finding the optimal path to collect them from the grid. The ‘**find\_nearest\_treasure**’ function in the code finds the closest treasure from the current position in the hex grid. If the current position is already a treasure, it returns the current position. Otherwise, it finds all the treasure locations within the hex grid and calculates the heuristic value (Manhattan distance) from current position to each treasure. The treasure with the lowest heuristic value is returned as the nearest treasure. This process is invoked every time the player moves to a new position on the grid to ensure that the path planning adapts to the player's current location, dynamically optimizing the route to the nearest uncollected treasure.

The ‘**a\_star**’ function implements the A\* search algorithm by using the cumulative cost and heuristic function to find the least step cost path to the nearest treasure. It initializes the search with the starting cell and sets up a priority queue (frontier) and an explored set. Using each node’s evaluation function *f(n) = g(n) + h(n)*, it iteratively explores cells by expanding and evaluating their neighbours. It applies the effects of traps and rewards encountered during the search and dynamically updates the goal to the nearest treasure. Energy cost calculations are managed separately and computed towards the end of this function as they do not influence the search algorithm directly.

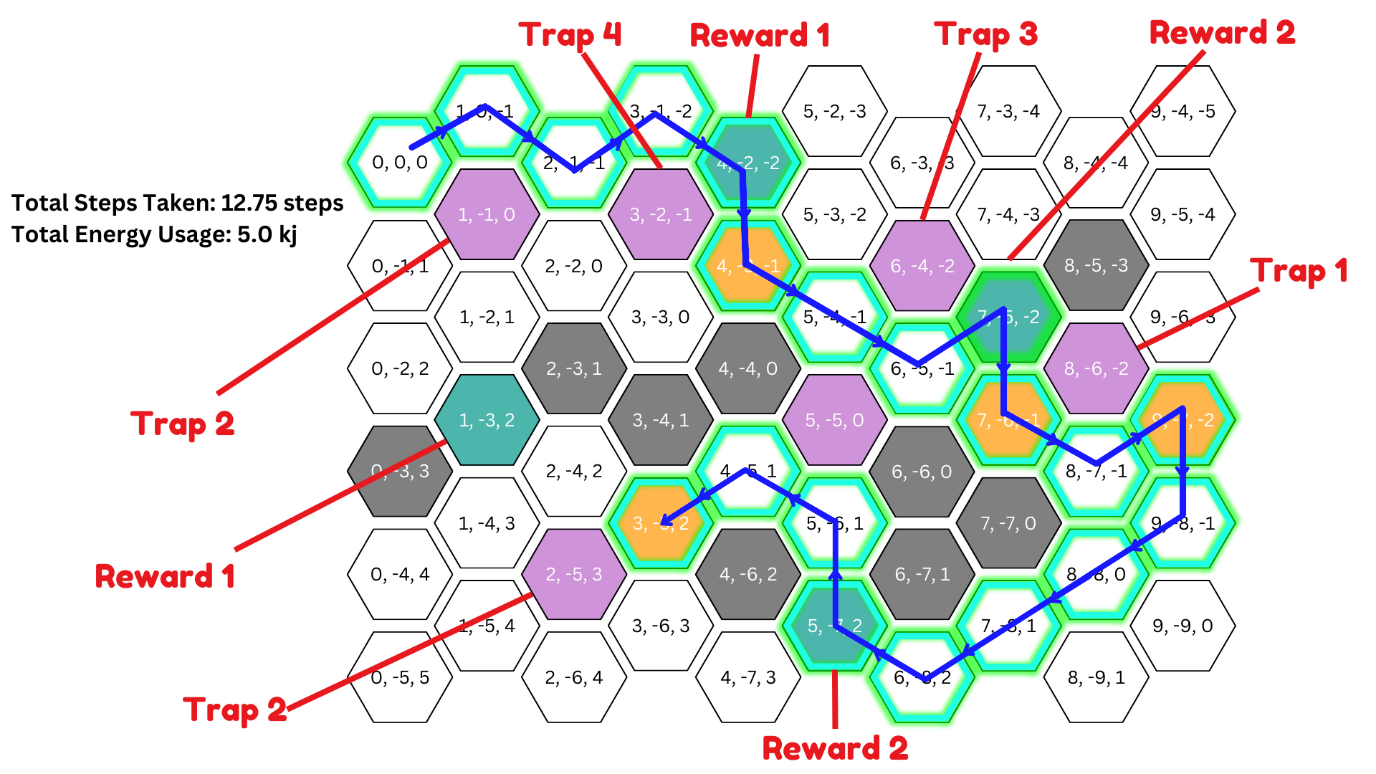
The ‘**treasure\_hunt**’ function is used to run the overall game. Starting from the initial position, the function repeatedly calls the ‘**a\_star**’function to find the shortest path to the nearest treasure. Each iteration involves updating total step costs, marking collected treasures, and adding the partial path to the *‘complete\_path’* list. The game continues until no more available paths to treasures are found or all treasures on the hex grid are collected. Lastly, the function returns the probable optimal complete path that leads to all goal nodes.

# **Solution Representation**

A white text with black and white text

Description automatically generated

**Figure 5.** Program Code Output



**Figure 6.** Visualization of the Code Output

From **Figure 5**, the algorithm begins from the starting cell (0,0,0), having taken a total of 0 steps and expended 0 energy, and concludes at cell (3, -5, 2), with a total of 12.75 steps taken and 5.0 kj of energy expended (rounded to 2 decimal places). It is shown that the program prioritizes treasure cells as primary targets, with reward cells being secondary targets encountered along the path to treasures. An exception occurs when a reward cell, such as at (6, -5, -1), is adjacent; in this case, the algorithm chooses to move to reward cell (7, -5, -2) instead of directly to the treasure cell (7, -6, -1) (**Figure 6**). It is also noted that the algorithm does not traverse across any traps in the map.

The algorithm’s strategic decision-making is evident in its avoidance of nearby traps at (1, -1, 0) and (3, -2, -1). For instance, it chooses to collect Reward 1 at cell (4, -2, -2) to avoid the trap at (3, -2, -1) while progressing towards the nearest treasure cell at (4, -3, -1). The algorithm reaches reward 1 (4, -2, -2) on its 4th step from the origin, having expended a total of 4 kJ of energy. Upon reaching this reward, it triggers the cell's effect, which initiates a continuous halving of the energy cost starting from the next step onwards. As the algorithm reaches the treasure cell (4, -3, -1) on its 5th step, it has garnered a total energy cost of 4.5 kJ and taken 5 steps (**Figure 5**).

The algorithm repeats the same process from where it left off, moving towards the consecutive nearby treasure cell (7, -6, -1). Along the way, it collects Reward 2 at (7, -5, -2), triggering a halving of the current step cost. As such, the algorithm reaches the second treasure cell at (7, -6, -1) with 8.5 steps taken and 4.97 kj energy expended (**Figure 5**).

The 3rd treasure is collected at (9, -7, -2) with 9.5 steps taken and 5.0 kj total energy expended (rounded). The algorithm proceeds to locate the 4th treasure at (3, -5, -2) while triggering Reward 2 at (5, -7, -2) along the way. Ultimately, the algorithm returns the solution path, having expended a total of 12.75 steps and 5.0 kJ of energy (rounded).

# **Algorithm Evaluation**

# **Performance and Efficiency**

The algorithm's use of cube coordinates for the hexagonal grid ensures accurate navigation and neighbour identification, which is crucial for implementing the A\* algorithm in such a grid structure. By representing the grid in three dimensions (x, y, z), the algorithm can handle the geometry of hexagonal grids effectively, providing a robust framework for accurate pathfinding.

The algorithm allows for a realistic simulation of varying traversal costs by reflecting real-time adaptations where conditions change as it progresses. It has a good handling of traps and rewards as it can adjust the *‘step\_cost’* based on traps and rewards encountered along the way, which allows the algorithm to adapt in real-time, ensuring that it continues to function effectively despite the changing conditions. For instance, encountering Reward 1 at (4, -2, -2) decreases the gravity of the world and every step’s energy cost is halved, the algorithm will modify the energy cost accordingly and maintain its efficiency throughout the path.

As for the heuristic function which was adapted specifically for hexagonal grids, ensures the A\* algorithm finds the shortest path by using a distance measure that suits the grid's geometry. This Manhattan distance heuristic which is tailored for hex grids can effectively narrow down the search space, making the pathfinding process more efficient. In the provided scenario, the algorithm quickly identifies paths to treasures, avoiding unnecessary cells and obstacles. Additionally, the heuristic function also contributes to a strategic pathfinding process as it has a prioritization of reward cells and avoidance of trap cells. It guides the algorithm toward beneficial cells and away from harmful ones unless absolutely necessary. An example of this would be of the algorithm choosing to take the path with Reward 1 at (4, -2, 2) to get to the treasure at (4, -3, -1) instead of going through paths (2, -2, 0) and (3, -3, 0) where there is nothing beneficial.

Another strength demonstrated by this algorithm is its dynamic goal search. By continuously searching for the nearest treasure, the algorithm ensures that it efficiently finds paths to all treasures. The algorithm effectively identifies and avoids obstacles, ensuring a valid path to the goal. For example, it reroutes around the obstacle at (4, -6, 2) to reach the treasure at (3, -5, 2). After collecting each treasure, the algorithm updates the goal to the next nearest treasure, ensuring efficient collection without redundant steps. Starting from (0, 0, 0), once a treasure is collected, the goal updates to the next nearest treasure, ensuring that the algorithm always targets the closest uncollected treasure.

By applying the A\* algorithm, it finds an optimal path that is both energy-efficient and step-efficient while collecting all treasures on the grid. This demonstrates how the algorithm can navigate through a dynamic environment while still being able to balance between step cost and energy cost. Overall, the algorithm's performance in accurate navigation, effective handling of traps and rewards, strategic path selection, and dynamic goal searching highlights its strengths and ensures it works as expected in the given scenario. The end output displays a well-balanced solution that meets the expected functionality, with a total of 12.75 steps taken and 5.0 kJ of energy used.

# **Limitations and Existing Issues**

The algorithm's complexity and performance present some limitations. Firstly, it has scalability issues. As the size of the hex grid and the number of traps and rewards increase, the algorithm's complexity grows which can potentially lead to performance issues in larger grids. The algorithm's performance might degrade due to the increased number of cells to evaluate and the dynamic goal updating mechanism. If the hex grid were to be doubled in size, the number of potential paths and the complexity of dynamic updates would significantly increase. This could potentially cause the algorithm to slow down because of the added computations.

Even though the algorithm can handle trap and rewards effectively, another limitation presented can be the inability to fully optimize complex interactions, such as encountering multiple traps or rewards consecutively. An example of a complex interaction would be the teleportation effect of Trap 3 as it can disrupt the planned path, leading to unpredictable behaviour if not properly managed. Ensuring that teleportation does not result in invalid or out-of-bound cells requires careful handling. For example, if Trap 3 at (6, -4, -2) moves the player to a distant cell which is not in the planned path direction, the algorithm must re-evaluate the entire path leading to a disruption in the flow. Similarly, Trap 4 at (3, -2, -1), which removes all uncollected treasures, could nullify the pathfinding efforts up to that point.

While the heuristic function's inclination towards rewards and avoidance of traps generally enhances efficiency, it can sometimes lead to suboptimal paths if it excessively prioritizes or avoids specific cells. Optimizing and fine-tuning the heuristic function is crucial to balance these biases and improve overall pathfinding efficiency. For instance, if a reward cell is significantly off the direct path, the algorithm might still choose it due to the heuristic bias, resulting in a longer route. Moreover, this biased search approach may not be universally suitable for all types of grid maps, emphasizing the need for nuanced adjustments to accommodate varying scenarios more effectively.

Finally, the algorithm's iterative technique to determining the nearest treasure and updating the path may not always result in the most optimal path for gathering all treasures, particularly in very complicated grids. Overall, while the algorithm has strong pathfinding capabilities, its limits reveal areas for additional optimization and refinement to increase performance in more complicated and bigger grid settings.