Contents

1.0 Problem Overview

An A* search algorithm is used for calculating the path, which determines the most economical path between the agent's current position to the nearest uncollected treasure. The purpose is to find the optimal method through a hexagonal grid area to get all the treasures. At the very start, the agent first identifies the starting node “entry” and then locates the treasure nodes to the hexagonal grid. Each iteration, the algorithm must select the closets treasure based on the heuristic function. The Euclidean distance between hex centres to determine the proximity of a prize. Once the target is chosen, the agent uses A* to compute a path that minimizes the total cost, which combines actual cost from start (g(n)) and estimated distance to goal (h(n)). As the agent follows the path node by node, it updates its total energy and steps using modifiers (multipliers) based on the type of each cell

2.0 State Space Representation

The graph's nodes, which correlate with the hex cells on the grid, shows each state in the search space while preserving the agent's current energy level, step count, and active multipliers that have an impact to the cost of movement, The environment itself is set up as a NetworkX graph. Each node is identified as its grid coordinate-row and column-as which carries extra data like attributes like their location and element type (e.g., 'entry', 'treasure', 'trap1', etc.). As a reflection of the hexagonal layout's connection, edges delineate legitimate transitions between neighbouring hex cells. Therefore, all accessible nodes in the graph creates the state space, and the cost and search path should be impacted by the traps and rewards received during the process.

The state is determined by several dynamic and contextual factors in addition to the agent's current location:

- current_position: Agent's current location on the grid.
- remaining_treasures: A dynamic set of yet-to-be-collected goals.
- energy_multiplier & step_multiplier: Adjust based on terrain effects (e.g., traps double energy cost).
- total_energy & total_steps: Cumulative resources consumed so far.

It needs to be recorded and maintained for more than just position, unlike static grid navigation issues, the environment here involves continuous state changes (such as increasing prices after traps). This makes an accurate planning and long-term performance evaluation.

3.0 Search Strategy: Why A* Search

A* is ideal here because it quickly shows the shortest route from the agents current spot to the closest treasure and, at the same time, weighs the non-uniform movement costs cause by traps and rewards. Instead of treating every step the same, A* combines the actual cost incurred so far, g(n) shaped by energy use and per-step multipliers), to a heuristic estimate , h(n), which in this case uses Euclidean distance between hex centres, to focus on paths that are both cost-effective and closer towards the goal. This heuristic fits a hex grid layout nicely, where distance is important, and algorithm dynamically adapts as the agent state changes. Situations like when multipliers are adjust by keying trap or reward cell or letting reactive and efficient pathfinding while collecting the treasure. A* is efficient and optimal when directed by a consistent heuristic, which is ideal for this dynamic pathfinding issue. It prioritize movements to the nearest treasures by using a Euclidean distance heuristic method, which nicely fits with the spatial aspect of the hex grid. As this method can handle weighted graph, it is able to account for the different movement costs by the rewards and trap. So instead of calculating one single global path, A* is being used iteratively between agents current location and the closest uncollected treasure. By using this technique it enables partial planning and real time adaptability in changing environment or the agents energy and step multiplier

4.0 Implementation Breakdown

4.1 A* Search Strategy

The A* algorithm is run iteratively. During each iteration we can use heuristic to select the nearest treasure. A* searches for the lowest-cost (in terms of adjusted step cost) path to the treasure. The agent then follows that path to update each step at each state. State changes could make a global plan invalid (e.g., a trap early in the path doubling future energy costs). By recalculating routes according to the present situation at each sub-goal, it permits partial planning.

4.2 Transition Model

When the agent moves from state S to a neighbour cell v, the new state S' is defined as follows:

- Position: updated to v
- Total Steps: incremented by 1 × step_multiplier
- Remaining Energy: decremented by 1 × energy_multiplier
- Multipliers:
    - If v is Trap1, then energy_multiplier *= 2
    - If v is Trap2, then step_multiplier *= 2
    - If v is Reward1, then energy_multiplier = max(0.5, energy_multiplier × 0.5)
    - If v is Reward2, then step_multiplier = max(0.5, step_multiplier × 0.5)
- Path History: append v to path
- Remaining Treasures:
    - If v is a treasure, remove it from the list

Special cells:

- Trap3: Teleports Ronny two steps back by popping the last two entries from the path_history, resetting the current position to that earlier point.
- Trap4: (Bug behavior) triggers a reset that mistakenly reassigns the treasure list to the already visited nodes, prematurely ending the loop.

4.3 State Representation

Each state in the search process is represented by:

- Position: the current cell (row, col) in the hex grid.
- Energy: the remaining energy budget.
- Steps: total steps taken (factoring in multipliers).
- Multipliers:
    - energy_multiplier: affects the energy cost per move.
    - step_multiplier: affects the step cost per move.
- Path History: a list of previously visited cells (used especially for trap behaviors).
- Remaining Treasures: a dynamic list updated as treasures are collected.

This encapsulates all variables affecting traversal decisions and future cost calculations. The grid is implemented as a networkx.Graph, with each hex cell as a node and edges between adjacent, walkable cells.

4.4 Heuristic Function

The heuristics works by measuring straight-line Euclidean distance between the centres of two hex cells, using the (x,y) pairs stored in each position. Returning that straight-line estimate helps A* use not only lower-cost nodes but also physically nearer to the target treasure, improving search efficiency on the hex grid.

4.5 Reconstruct_path

Once A* picks the goal node , reconstruct_path traces backward through the came_from dictionary, which records each explored node best predecessor. It then produces the list order of nodes the reflect the best path by constantly the links from objective back to the beginning and then reversing the order

4.6 plot_path.

Using **plot_path**, the calculated path overlays over the hex grid plot to display the results. At each node point along the way, orange circles are drawn after any previous path marks have been removed. The path is prominently shown against the grid's-coloured hex cells thanks to this colour and design decision.

4.7 Treasure Collection Loop (collect_all_treasures)

The agent's state is initialised in the main loop by setting the energy to 100, the steps to 0, and the energy and step multipliers to 1.0. The loop then repeats until all of the riches have been gathered or there are no more viable paths.

In each iteration, it selects the nearest uncollected treasure (by heuristic), runs A* to get a subpath, and then traverses that subpath node by node.

The agent manages unique behaviours (such as teleporting back two steps on Trap3 or finishing treasure gathering too soon on Trap4), modifies multipliers for rewards and traps, updates energy and step counts, and adds each visited node to the overall route as it enters each cell.

4.8 Grid and Graph Representation

The hexagonal grid is drawn with Matplotlib by placing flat-topped hexagons at positions determined by custom horizontal and vertical spacing. Underneath the visualization, the environment is showed in a NetworkX graph: each hex cell is a node labeled with its element type (entry, treasure, trap, reward, obstacle), and edges connect adjacent, non-obstacle cells. Edge weights correspond to the penalties of entering the destination cell.

4.9 Movement Effects

Entering trap or reward cells adjusts future movement costs via multiplicative effects:

- Trap1 doubles the energy cost
- Trap2 doubles the step count
- Reward1 halves the energy multiplier (down to 0.5)
- Reward2 halves the step multiplier (down to 0.5).
- Trap3 teleports the agent two steps back along its path history
- Trap4 triggers a (buggy) reset of remaining treasures

5.0 Assumptions and Design Choices

A number of assumptions were adopted in this solution to help ease the problem of maze environment:

- Ronny begins at the entry cell, and treasures have to be received in the most efficient order to save energy and steps.
- The environment can be viewed completely, i.e. the coordinates of every treasure, every trap, and every reward are available at the start.
- The hexagonal maze is considered as an undirected graph with node (room) and edge (valid movement (not an obstacle)).
- Movement cost is determined by the penalty of destination cell. The obstacles are not included in the graph, and they are infinitely costly.
- A Search* was selected because of its optimality and the heuristic employed is appropriate to the spatial structure of the maze.
- The impact of the traps and rewards is implemented when a particular cell is entered. As an instance, reward1 reduces energy use by half, whereas trap2 increases the cost of a step by twice as much.

6.0 Research Objectives & Evaluation Metrics

The aim of this project is to simulate an intelligent pathfinding algorithm which will help Ronny to collect all the treasures in a confined maze taking into consideration energy, movement penalties and special tile effects. It is not only about reaching the treasures but reaching them efficiently and smartly.

The solution is evaluated using the following metrics:

- Total Steps – Counts Ronny's movements, adjusted by step multipliers.
- Remaining Energy – Shows how efficiently Ronny moved through the maze.
- Unique Nodes Visited – Checks if Ronny avoided unnecessary revisits.
- Goal Accuracy – Confirms all treasures were collected within limits.

.

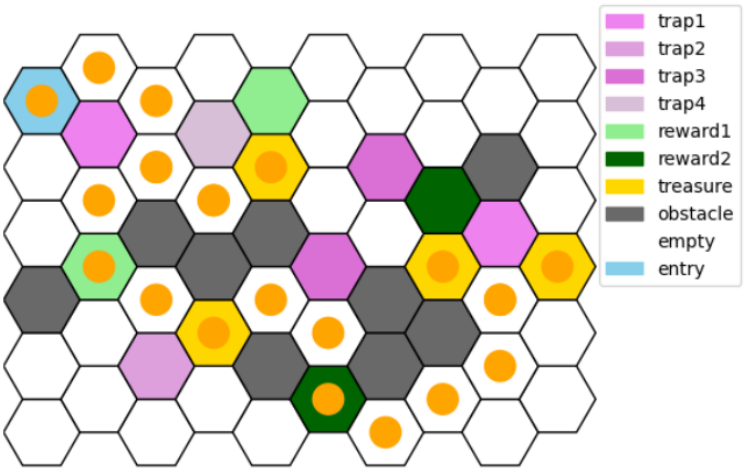
7.0 Conclusion

A* search algorithm was able to help Ronny navigate through the maze and pick all the treasures available. The algorithm found the most efficient route even in the presence of various traps, obstacles and dynamic movement penalties, by prioritizing the closest reachable treasure and recomputing the route whenever a modifier such as a reward or trap was reached.

Important tile effects like Reward1 (energy cost is halved) and Reward2 (step cost is halved) were implemented in a way that they went hand in hand with the traversal logic, which contributed to the optimization of Ronny. Trap3, which enforces a backward teleport, and Trap4, which re-consider treasure targets, were also treated as they were supposed to be, and the integrity of the pathfinding process was maintained.

The final result shows that all four treasures were acquired using the least amount of energy and only 17.5 steps taken. The program has shown to be complete and flexible, proving A* to be a very effective and suitable algorithm in the solving of constrained, dynamic pathfinding problems such as this one.

8.0 Result display/visualization



The final outcome is displayed in a hexagonal grid map with clear labels, and each tile is clearly color-coded and labelled with easy-to-understand symbols indicating its type:

- Traps – Purple tiles
- Rewards – Light green (Reward1) and dark green (Reward2)
- Treasures – Gold tiles
- Entry – Sky blue tile

The complete route of Ronny is clearly marked by orange circles that are drawn on the tile that he visited. This gives one a straightforward way to track his progress in making his way through the maze, how he avoided obstacles and traps and how he used rewards to minimize the overall cost. The design provides a visual support of the effectiveness of the search plan and depicts the ability to make adaptive choices during the journey.

Summary of Results:

- Final steps taken: 17.5
- Remaining energy: 85.0
- Treasures collected: 4
- Unique nodes visited: 19