

Assignment 1

2024 April Semester

CSC3206 Artificial Intelligence

NAME: LESTER KOON

STUDENT ID: 20068813

Question

Treasure hunt in a virtual world

You are organizing a treasure hunt in a virtual world filled with obstacles, traps, and treasures. When you enter the virtual world, to travel from one cell to another cell, you will take exactly one step. You must navigate through the virtual world to collect the treasures while avoiding traps. Activated traps will increase the difficulty to explore the world while rewards are available to ease the exploration.

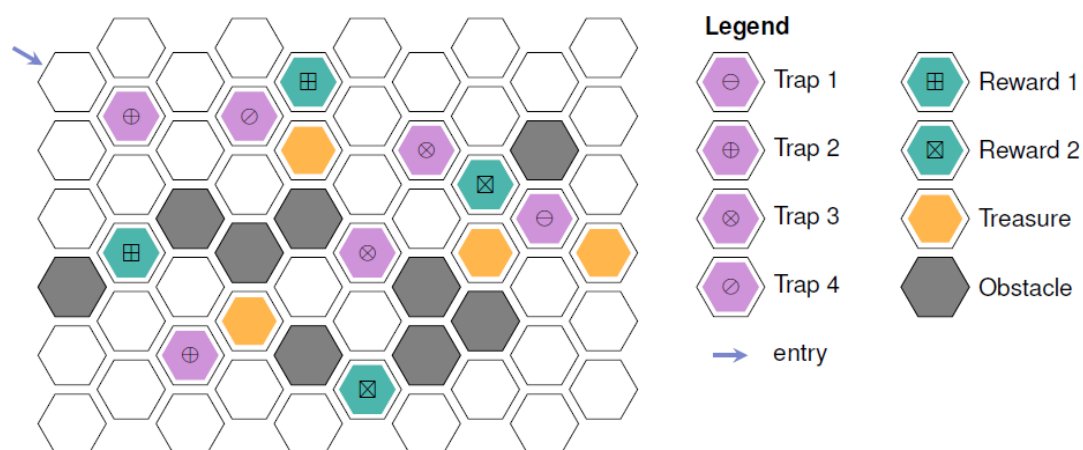








Figure 1: Map of the virtual world

The effect of the traps and rewards are described in Table 1.

Table 1: Descriptions for Traps and Rewards

	Trap 1	This trap will increase the gravity of the world. Every step you take will consume double the energy as previous.
	Trap 2	This trap will decrease your speed. You will take double the steps to move to the adjacent cell.
	Trap 3	This trap will move you two cells away following your last movement direction.
	Trap 4	This trap will remove all treasures that have not been collected. All treasures that are collected will not be affected.
	Reward 1	This reward will decrease the gravity of the world. Every step you take will consume half the energy as previous.
	Reward 2	This reward will increase your speed. You will take half the steps to move to the adjacent cell.

Problem Analysis

Components of the map:

- Empty Cell: A cell that can be freely traversed
- Obstacle: A cell that cannot be traversed
- Trap: A cell that negatively affects the agent's objective
- Reward: A cell that positively affects the agent's objective
- Treasure: A cell containing a treasure that the agent must collect

The overall view of the virtual map, and the details of all the traps and rewards can be found in the figures above.

Initial State

The starting position of the agent in the map, where no treasures have been collected yet.

Goal State

The goal state is achieved when the agent when the agent effectively acquires all the treasures that are currently located on the map. If all treasures have been obtained and there are no treasures present on the map, the agent's position at the goal state can be anywhere in the grid and is not fixed.

Path Cost

The path cost is incurred by the agent to traverse from the initial state to the goal state.

- Energy Cost: The amount of energy consumed by the agent.
- Step Cost: The number of steps taken by the agent.

Since there is no difference between the energy and steps cost (both have a default value of 1), the path cost is a sum of both of the cost together.

State Representation:

The state space encompasses all possible configurations of the agent's position, represented by a cell in the map with coordinates (x, y), path traversed and current path cost.

Transition Model

The transition model describes the result of actions that the agent takes:

- Empty Cell: The agent moves with normal movement cost.
- Obstacle: The agent's movement to an obstacle cell is not possible.
- Trap: The agent activates a following effect:
 - \ominus Trap 1: Doubles energy consumption for subsequent moves.
 - \oplus Trap 2: Doubles the number of steps required for subsequent moves.
 - \otimes Trap 3: Moves the agent two cells away in the direction of the last move.
 - \bigcirc Trap 4: Removes all uncollected treasures from the grid.
- Reward: The agent activates a following effect:
 - \boxplus Reward 1: Halves energy consumption for subsequent moves.
 - \boxtimes Reward 2: Halves the number of steps required for subsequent moves.
- Treasure: The agent collects the treasure and might achieve the goal.

Solution Algorithm

Selected Algorithm

A* Search Algorithm

Code Implementation

Program Flowchart

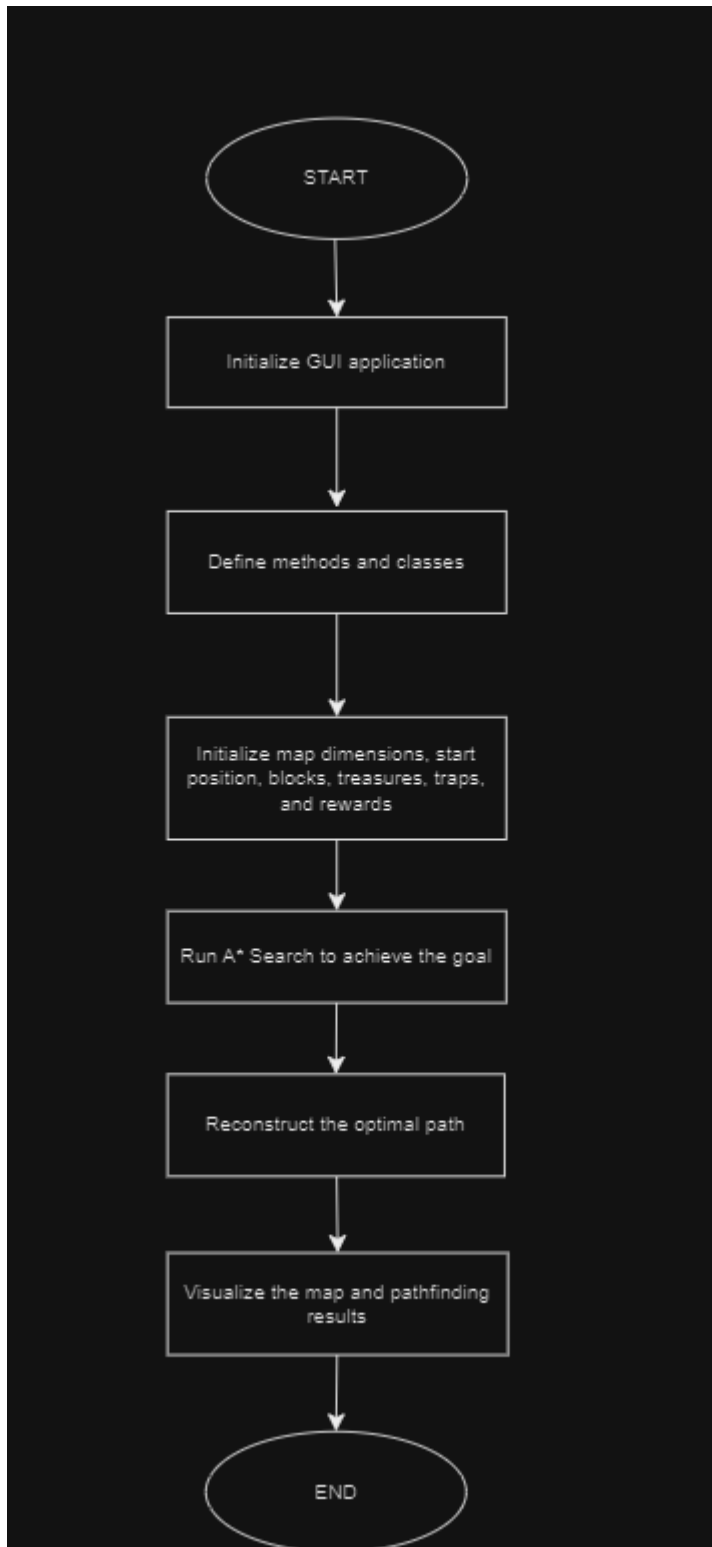


Figure 11 Program Flow

Results and Output

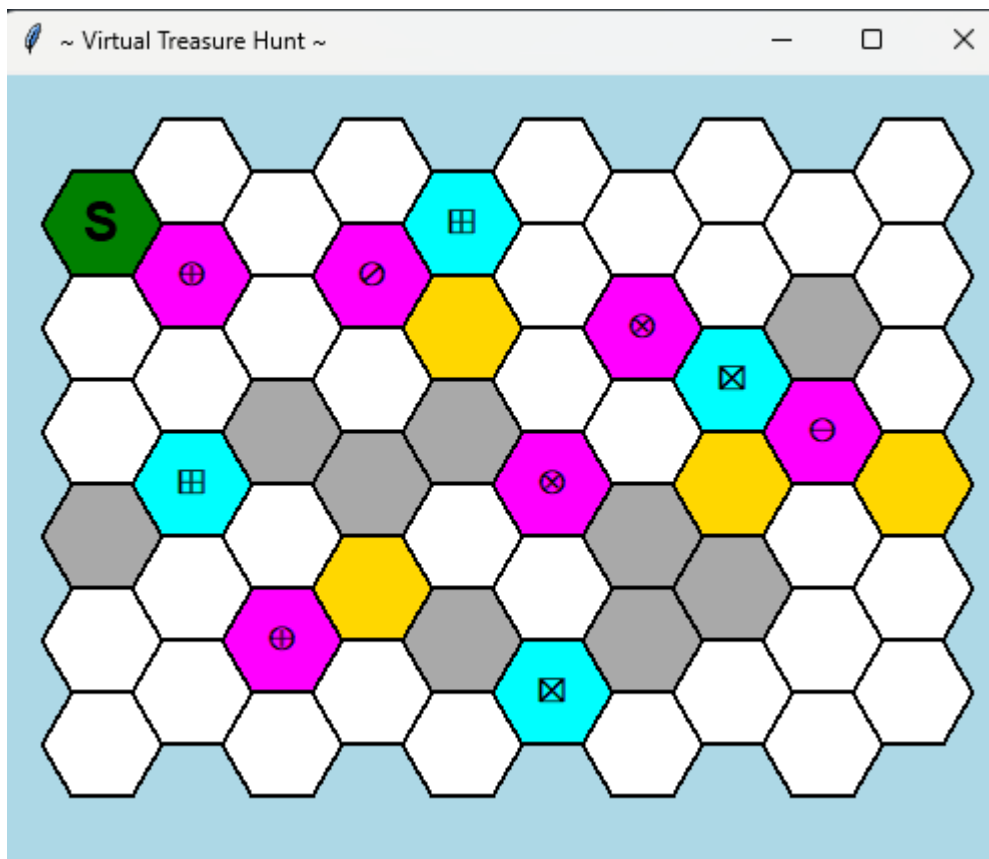


Figure 2 Initial Map Visualization

Above is the recreated map of the virtual treasure hunt world. This helps the users visualize the dimensions of the map, and the position of elements such as traps, treasures and rewards.

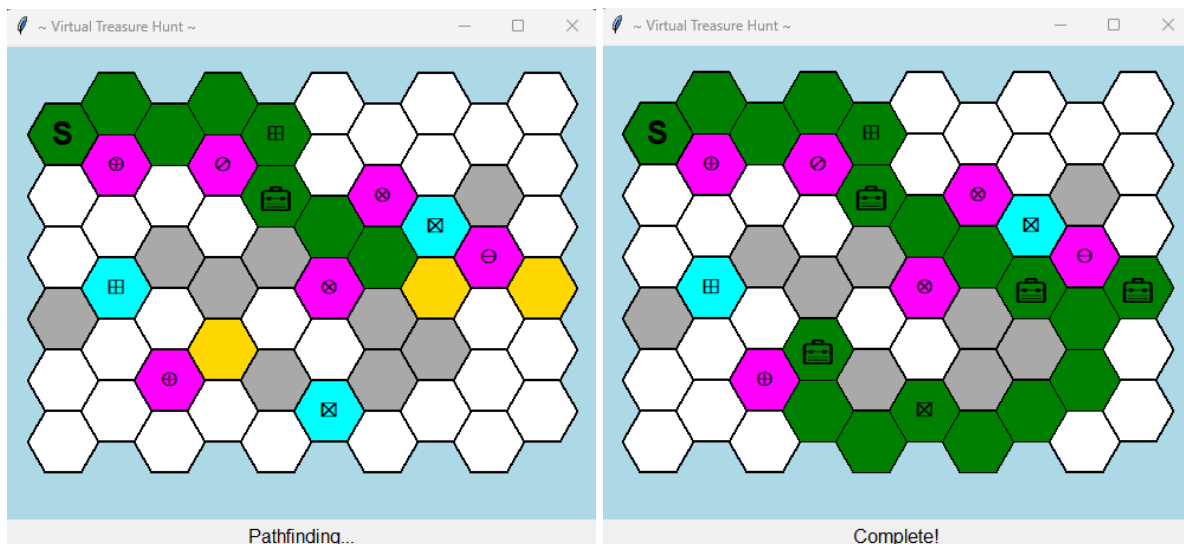


Figure 32 Pathfinding Visualization

The figures above show the visualization of the optimal path discovered by the A* search algorithm, step-by-step.

```
Virtual Treasure Hunt using A* search

-----
Goal: Collect all the treasures with the optimum route
-----

END!
You have reached the goal.
Path:
[(1, 0), (2, 0), (3, 0), (4, 0), (4, 1), (5, 2), (6, 2), (7, 3), (8, 3), (9, 3), (8, 3), (8, 4), (7, 5), (6, 5),
(5, 5), (4, 5), (3, 5), (3, 4)]
Total path cost: 18
```

Figure 4 Terminal Interface

Figure 4 shows the terminal interface of the program, it give the brief function of the program and also shows the detailed results of the path found.

Implementation Explanation

For simple an small maps, the A* search algorithm is a very effective pathfinding and algorithm used to find the shortest path from a start node to a goal node, by using a combination both Greedy Best-First-Search and Dijkstra's algorithm. Greedy Best-First-Search uses a heuristic to estimate the cost from the current node to the goal, helping A* prioritize nodes that seem closer to the goal and making the search more focused and efficient. Meanwhile, Dijkstra's Algorithm ensures the shortest path by considering all possible paths and updating the cost estimates for each node, with A* integrating

this by evaluating the total cost ($f = g + h$), where g is the actual cost from the start and h is the heuristic estimate.

Heuristic Function

The heuristic function estimates the cost from the current cell to the nearest goal by help prioritizing the nodes in the search by estimating the remaining cost to reach a goal. The function selected for the program is Euclidean distance as shown below:

$$d = \sqrt{[(x_2 - x_1)^2 + (y_2 - y_1)^2]}$$

Figure 5 Euclidean distance formula

Where,

- d is Euclidean Distance
- (x_1, y_1) is the current cell position
- (x_2, y_2) is the next cell position

A* Search Strategy

It combines the real cost, g , with the heuristic estimate, h . The total cost function, f , can be defined as follows:

$$f = g + h$$

All nodes are inserted into a priority queue. This sort the next possible node in relation to their ability to reach the goal, which ensures that the node with the minimum total cost, f , is always expanded.

A* Search Flow

For each treasure in the list:

1. Define the start node and goal:

Start Node: This is where the search begins. Initially, it's the starting position of the search. After finding a treasure, the start node is updated to the position of the last found treasure.

Goal Node: This is the current treasure we are trying to reach. It's set as the destination for this particular A* search.

2. Initialize A* Search Data Structures:

Priority Queue: This helps keep track of nodes that need to be evaluated.

The start node must be inserted to the queue with its initial f value, which is calculated as $f = g + h$.

- g is the cost from the start node to this node.
- h is the heuristic estimate from this node to the goal.
- f is the sum of g and h, representing the total estimated cost to reach the goal via this node.

Closed List: This list keeps track of nodes that have already been evaluated to avoid reprocessing.

Main Search Loop

While there are nodes still treasure:

1. Extract Node:

Remove the node with the lowest f value from the open list. This node is the most promising candidate for evaluation based on the cost estimates.

2. Check Goal:

If the extracted node is identified as the goal, which in this case is the current treasure, several steps are taken. First, rebuild the path by tracing back from the goal node to the

start node using the parent references of each node. This process outlines the shortest route taken to reach the treasure. Second, record this path as the shortest route to the treasure found. Update the current position to the location of this newly found treasure. Remove this treasure from the list of remaining treasures in order not to find the same thing again. Finally, exit the loop as the goal has been reached, and no further processing is needed for this treasure.

3. Generate Neighbours:

For each neighbour of the current node, several checks and calculations are performed. First, it checks if this is a valid neighbour and has not gone out of the grid boundaries, is not blocked, or occupied. After that, calculate the costs associated with moving into that neighbourhood. Update the neighbour's g , h , and f values if the neighbour is not already in the closed list or if a cheaper path is found. Set the neighbour's parent to the current node to facilitate path reconstruction later. Finally, add the neighbour to the open list if it is not already present, ensuring it is considered for future evaluation. Once all neighbours of the current node have been evaluated, move the current node to the closed list. This step marks the node as processed and prevents it from being re-evaluated in subsequent iterations.

Path Reconstruction and Update

Once the goal is reached, the algorithm reconstructs the path by tracing back from the goal node to the start. This process of comprises of tracing the parent links from the destination node to its parent node till reaching the start node, thus forming the complete path that led to the treasure. The reconstructed path signifies the most efficient route discovered by the A* algorithm. After the path is reconstructed, the current position is updated to the location where the treasure was found. Following this, the found treasure is removed from the list of treasures to ensure that the search continues for the remaining treasures.

Repeat for Remaining Treasures

- Continue Search: Perform the A* search again for the next treasure in the updated list of remaining treasures, starting from the current position (the last found treasure).
- Update Start Position: The start position is updated to the location of the most recently found treasure.
- Process Next Treasure: Repeat the A* search process to find the shortest path to the next treasure.

Evaluation and Discussion

Does it work as expected?

The A* algorithm is designed to find the shortest path from the start node to the goal node. In the context of collecting multiple treasures, the algorithm correctly identifies the shortest path to each individual treasure that is nearest to the current node. Thus, by treating each treasure as an independent goal ensures accuracy in pathfinding and the shortest route. After, cross-referencing with human trials, the result of the programs is equal, if not better than the possible solutions created by humans.

Advantages

- Able to handle map complexity and dynamic elements:
 - The algorithm and heuristic used in the program was able to find the optimal path despite the complexity of the map which was scattered with traps and obstacles and was able to successfully avoid them. Furthermore, with the incorporation of dynamic constraints caused by the traps' and rewards' effects, the program was still able to handle them and effectively navigate through traps while leveraging rewards to optimize the path.
- Able to handle multiple goals

- The current approach utilizes sequential processing by breaking the goal to find all treasures, into smaller, more simplified segments. This enables multiple goals to be set in the same map, as each goal is tackled independently, allowing the search process to be refined and adapted as the position changes throughout the path.
- Flexibility in heuristic function
 - The choice of heuristic function such as Manhattan Distance, Euclidean Distance, or other domain-specific heuristics, allows the algorithm to be tailored to specific map characteristics and objectives. This makes the heuristic flexible enough to be tuned in order to better reflect the nature of problem.

Limitations

- Not suitable for larger maps
 - As the size of the map and complexity increases, the performance of the program will degrade due to the limitations of the A* search structure. The open list can grow substantially, leading to increased memory consumption and slower processing times.
- Effectiveness is dependent on the heuristic function
 - The heuristic for each A* search must be carefully picked and implemented according to the characteristics of the problem and the program's structure. A poorly chosen heuristic may lead to inefficient search paths, which will be more significant for larger, more complex maps.
- Suboptimal paths due to sequential processing
 - By addressing each treasure individually, the algorithm may not always find the globally optimal path that considers all treasures collectively. However, the problem of finding a globally optimal route that minimizes the total travelled distance for all the treasures is much more complex, usually requiring advanced techniques or algorithms beyond basic A*.