ARTIFICIAL INTELLIGENCE

MINI-PROJECT

REGISTER NUMBER	2117240070342
NAME	VASUKI K
PROJECT TITLE	AI Treasure Hunt
DATE OF SUBMISSION	
FACULTY IN-CHARGE	Mrs. S. Divya

Signature of Faculty in Charge

TREASURE HUNT AI GAME

PROBLEM STATEMENT

Artificial Intelligence (AI) enables systems to simulate human-like decision-making in problem-solving environments. In this project, an AI agent navigates a grid-based environment to find hidden treasure while avoiding traps. The system applies AI search algorithms to simulate intelligent behavior within a constrained environment. The goal is to demonstrate how an agent can autonomously find an optimal path to a target using heuristic search, representing an application of AI in pathfinding and decision-making.

THEORETICAL BACKGROUND

Artificial Intelligence (AI) is the science of creating systems capable of performing tasks requiring human-like intelligence. In grid-based pathfinding problems, AI uses search algorithms to explore the environment efficiently. This project uses the A* (A-star) algorithm, a popular AI pathfinding technique that combines the advantages of Dijkstra's algorithm and Greedy Best-First Search. A* uses a heuristic function to estimate the cost of reaching the goal, allowing the agent to find the shortest and safest path to the treasure while avoiding obstacles.

JUSTIFICATION FOR CHOOSING THE ALGORITHM

The A* Search Algorithm was chosen for its balance between optimality and efficiency. It is one of the most effective pathfinding algorithms for deterministic environments like the Treasure Hunt grid. Reasons for selection include:

• Optimal and complete solution – A* always finds the best path if one exists.
• Efficient computation – It prunes unnecessary paths using heuristics.
• Adaptability – Can handle different grid sizes and obstacle placements.
• Educational value – Demonstrates core AI search and heuristic concepts clearly.

IMPLEMENTATION AND CODE

The project was implemented in Python using the Tkinter library for GUI representation. The grid represents the environment with the agent, traps, and treasure. The A* algorithm computes the optimal route for the agent from the start position to the treasure by minimizing the cost function f(n) = g(n) + h(n), where g(n) is the path cost and h(n) is the heuristic (Manhattan distance).

CODE :

```
import tkinter as tk

import random

from tkinter import messagebox

import heapq


SIZE = 5

CELL_SIZE = 80


class TreasureHuntAI:

    def __init__(self, root):

        self.root = root
```

```
        self.root.title("AI Treasure Hunt 🤖💰")

        self.canvas = tk.Canvas(root, width=SIZE * CELL_SIZE, height=SIZE
* CELL_SIZE, bg="white")
        self.canvas.pack()

        self.reset_button = tk.Button(root, text="Restart Game", font=("Arial",
12, "bold"),
                                      command=self.reset_game, bg="#4CAF50",
fg="white")
        self.reset_button.pack(pady=10)

        self.reset_game()

    def reset_game(self):
        self.grid = [["-" for _ in range(SIZE)] for _ in range(SIZE)]
        self.agent = [0, 0]
        self.score = 0

        # Place traps
        for _ in range(3):
            x, y = random.randint(0, SIZE - 1), random.randint(0, SIZE - 1)
            if [x, y] != [0, 0]:
                self.grid[x][y] = "X"
```

```
        # Place treasure
        while True:
            tx, ty = random.randint(0, SIZE - 1), random.randint(0, SIZE - 1)
            if self.grid[tx][ty] == "-":
                self.grid[tx][ty] = "T"
                self.treasure = [tx, ty]
                break

        self.draw_grid()
        self.path = self.a_star_path()
        if not self.path:
            messagebox.showinfo("Result", "No safe path to treasure! 😢")
        else:
            self.root.after(1000, self.move_agent)

    def draw_grid(self):
        self.canvas.delete("all")
        for i in range(SIZE):
            for j in range(SIZE):
                x1, y1 = j * CELL_SIZE, i * CELL_SIZE
                x2, y2 = x1 + CELL_SIZE, y1 + CELL_SIZE
```

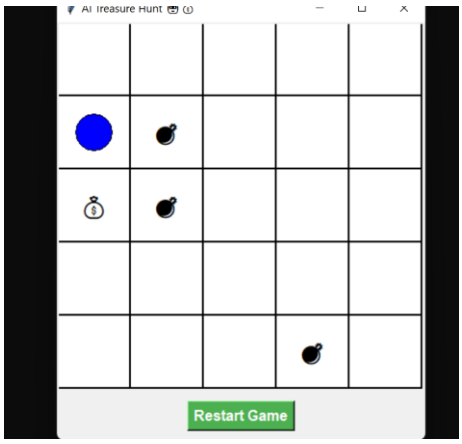
```
            self.canvas.create_rectangle(x1, y1, x2, y2, outline="black",
width=2)


            cell = self.grid[i][j]
            if [i, j] == self.agent:
                self.canvas.create_oval(x1 + 20, y1 + 20, x2 - 20, y2 - 20,
fill="blue")
            elif cell == "X":
                self.canvas.create_text(x1 + CELL_SIZE / 2, y1 + CELL_SIZE
/ 2, text="💣", font=("Arial", 22))
            elif cell == "T":
                self.canvas.create_text(x1 + CELL_SIZE / 2, y1 + CELL_SIZE
/ 2, text="💰", font=("Arial", 22))


    def is_valid(self, x, y):
        return 0 <= x < SIZE and 0 <= y < SIZE and self.grid[x][y] != "X"


    def heuristic(self, a, b):
        # Manhattan distance
        return abs(a[0] - b[0]) + abs(a[1] - b[1])


    def a_star_path(self):
        """Finds shortest safe path to treasure using A*"""
        start = tuple(self.agent)
```

```
goal = tuple(self.treasure)
moves = [(0,1),(1,0),(-1,0),(0,-1)]

open_list = []
heapq.heappush(open_list, (0, start))
came_from = {start: None}
g_score = {start: 0}

while open_list:
    _, current = heapq.heappop(open_list)

    if current == goal:
        path = []
        while current:
            path.append(current)
            current = came_from[current]
        return path[::-1]  # reverse path

    for dx, dy in moves:
        nx, ny = current[0] + dx, current[1] + dy
        neighbor = (nx, ny)
        if self.is_valid(nx, ny):
            tentative_g = g_score[current] + 1
```

```
                if neighbor not in g_score or tentative_g < g_score[neighbor]:
                    g_score[neighbor] = tentative_g
                    f_score = tentative_g + self.heuristic(neighbor, goal)
                    heapq.heappush(open_list, (f_score, neighbor))
                    came_from[neighbor] = current
        return None

    def move_agent(self):
        if not self.path:
            messagebox.showinfo("Result", "No path found!")
            return

        if len(self.path) > 1:
            self.path.pop(0)
            self.agent = list(self.path[0])
            self.draw_grid()
            self.root.after(700, self.move_agent)
        else:
            messagebox.showinfo("🎉 Victory!", "AI found the treasure successfully!")

# Run
root = tk.Tk()
```

```
app = TreasureHuntAI(root)

root.mainloop()
```

OUTPUT AND RESULTS



LITERATURE SURVEY :

Pathfinding is a core problem in Artificial Intelligence where an agent must reach a goal while avoiding obstacles. The *AI Treasure Hunt* project applies this concept on a grid with traps and a treasure, using the **A*** algorithm for efficient search.

A* (Hart et al., 1968) combines actual path cost g(n)g(n)g(n) and heuristic estimate h(n)h(n)h(n) to find optimal routes. When hhh is admissible (e.g., Manhattan distance), A* guarantees the shortest path while exploring fewer nodes than **BFS** or **Dijkstra's** algorithm.

Heuristics such as **Manhattan**, **Euclidean**, or **Diagonal** distances guide the search; the Manhattan heuristic fits 4-directional grid movement used here. The algorithm uses a **priority queue** to expand nodes with the lowest total cost f(n)=g(n)+h(n)f(n)=g(n)+h(n)f(n)=g(n)+h(n).

Improved variants include **Weighted A*** (trades optimality for speed), **IDA*** (memory-efficient), **D*** and **D* Lite** (dynamic environments), and **Jump Point Search (JPS)** for faster grid traversal. These are used in robotics, navigation, and games.

REFERENCES

1. Russell, S., & Norvig, P. (2021). Artificial Intelligence: A Modern Approach. Pearson.
2. Amit Patel, 'Introduction to the A* Algorithm', Red Blob Games https://www.redblobgames.com/pathfinding/a-star/

3. Python Tkinter Documentation https://docs.python.org/3/library/tkinter.html

4. AI Pathfinding Examples on GitHub – Open Source Educational Projects.

https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf?utm_source=chatgpt.com