Smart Maze Game – AI-Based Pathfinding Project Report

1. Project Overview

The Smart Maze Game is a Python application designed to demonstrate AI-based pathfinding and decision-making. It features two versions:

- **GUI Version:** Uses Pygame to generate and solve a random maze with various AI algorithms.
- **Terminal Version:** Models an agent navigating a grid with obstacles using several AI algorithms to optimize movement and decisions.

2. Technologies Used

- Python
- Pygame (GUI & graphics)
- NumPy (array handling)
- Scikit-learn (for SVM)
- Heapq (used in A* algorithm)

3. GUI Version: Smart Maze Game

How It Works

- Generates a random 20x20 maze with walls and empty cells.
- Start cell at top-left (0,0); goal at bottom-right (19,19).
- User selects AI algorithm via buttons to solve the maze.
- Displays the solution path graphically with color-coded cells.

Algorithms Used (currently placeholders use A*)

Algorithm	Description
A* Search	Finds shortest path using Manhattan heuristic.
PSO	for Particle Swarm Optimization.
ACO	for Ant Colony Optimization.
SVM	for Support Vector Machine logic.
Revolutionary	for evolutionary algorithm.
Perceptron	for neural network logic.

GUI Features

- Maze displayed as grid:
  - White: empty cell
  - Black: wall
  - Green: start
  - Red: goal
  - Purple: path
- Buttons to run algorithms or reset maze.
- Displays number of steps in solution path.

Code Structure

Component	Role

Button class	Handles buttons and click events.
generate_maze	Creates maze with random walls.
a_star	Implements A* search.
solve_with_animation	Animates solution path step-by-step.
draw_maze	Draws maze and solution on screen.
main function	Controls game loop and user interaction.

4. Terminal Version: AI Agent on Grid

Environment

- 10x10 grid with free (0) and blocked (1) cells.
- User inputs obstacle positions, target coordinates, agent health, and iterations for optimizations.
- Agent starts at (0,0) aiming to reach target.

AI Algorithms

Algorithm	Purpose
Particle Swarm Optimization (PSO)	Finds optimal speed and angle for fastest travel.
Ant Colony Optimization (ACO)	Finds shortest path avoiding obstacles.
Support Vector Machine (SVM)	Decides whether to attack or retreat based on health & distance.
Revolutionary Algorithm	Evolves strategy weights balancing movement and obstacle avoidance.
Perceptron	Decides whether to pursue target based on features.

Details

- PSO: Particles search speed & angle to minimize travel time + angle penalty.

- ACO: Uses pheromone trails and heuristics for path optimization.
- SVM: Binary classifier trained on sample data for attack/retreat decision.
- Revolutionary Algorithm: Evolutionary process with selection, crossover, mutation.
- Perceptron: Simple neural model trained over epochs for pursuit decision.

5. User Interaction & Outputs

User inputs:

- Target coordinates
- Number and positions of obstacles
- Agent health (0 to 1)
- Number of iterations for PSO, ACO, etc.

Outputs:

- PSO: Best speed and angle.
- ACO: Best path or failure message.
- SVM: Recommended action (attack/retreat).
- Revolutionary: Best strategy weights.
- Perceptron: Pursue decision (yes/no).

6. Strengths & Possible Improvements

Strengths

- Integrates diverse AI methods: search, optimization, machine learning, evolutionary.
- Modular design with clear separation of algorithms and environment.
- Realistic constraints for grid navigation and decision-making.
- Visual and terminal versions complement each other.

Improvements

- Implement real PSO, ACO, SVM, Perceptron, and Revolutionary algorithms in GUI version (currently placeholders).
- Add difficulty levels and maze customization.
- Enhance fitness functions to account for obstacles explicitly.
- Visualize terminal grid and algorithm results.
- Expand training datasets for SVM and Perceptron for better generalization.
- Integrate algorithms to influence one another (e.g., PSO tuning ACO parameters).

7. Learning Outcomes

- Understood A* search and AI pathfinding in maze environments.
- Gained experience visualizing AI steps using Pygame.
- Learned integration of optimization and machine learning algorithms in decision-making.
- Built a flexible, expandable AI project framework.