**What This Project Does (In Simple Terms)**

Imagine a warehouse floor shown as a grid. Your robot wants to go from a **start point** to a **goal point**, but there are obstacles in the way. You need a smart way for it to find the shortest path around the obstacles.

This project simulates that — using the **A\*** (A-Star) algorithm to find the best path, and then moves the robot step-by-step while showing its position on the grid.

**🔍 Project Overview**

Here's what the program is made of:

css

CopyEdit

robot-pathfinding-simulator/

├── main.py ← Starts the simulation

├── grid.py ← Represents the warehouse/grid

├── robot.py ← Represents the robot and its movement

├── pathfinding/

│ └── a\_star.py ← The A\* algorithm that finds the best path

└── visualization/

└── logger.py ← Prints the grid and robot position

**📄 1. grid.py — The Warehouse World**

**Think of it like:**

A map with walls (obstacles) and open areas.

**What it does:**

* Defines the size of the grid
* Knows where obstacles are
* Gives the robot valid places it can move to

python

CopyEdit

class Grid:

def \_\_init\_\_(self, width, height, obstacles):

self.width = width

self.height = height

self.obstacles = set(obstacles) # Places the robot cannot go

def in\_bounds(self, pos): # Is the position inside the grid?

x, y = pos

return 0 <= x < self.width and 0 <= y < self.height

def passable(self, pos): # Is the position NOT an obstacle?

return pos not in self.obstacles

def neighbors(self, pos): # Where can the robot go next?

x, y = pos

neighbors = [(x+1, y), (x-1, y), (x, y+1), (x, y-1)] # 4 directions

return [p for p in neighbors if self.in\_bounds(p) and self.passable(p)]

**📄 2. robot.py — The Robot**

**Think of it like:**

A character that can move one step at a time.

**What it does:**

* Knows where it is now (position)
* Follows a list of steps (path) to the goal
* Moves one step when asked

python

CopyEdit

class Robot:

def \_\_init\_\_(self, grid, start, goal):

self.grid = grid

self.position = start

self.goal = goal

self.path = []

def set\_path(self, path): # Give the robot a route to follow

self.path = path

def move\_step(self): # Move one step along the path

if self.path:

self.position = self.path.pop(0)

**📄 3. pathfinding/a\_star.py — The Brain**

**Think of it like:**

Google Maps for the robot — it finds the fastest route.

**A\* Algorithm:**

* Looks at how far each move gets you to the goal
* Avoids going through walls
* Picks the smartest path

python

CopyEdit

def heuristic(a, b): # Distance from a to b

return abs(a[0] - b[0]) + abs(a[1] - b[1])

def a\_star\_search(grid, start, goal):

# Priority queue: (priority, position)

frontier = [(0, start)]

came\_from = {start: None}

cost\_so\_far = {start: 0}

while frontier:

\_, current = heapq.heappop(frontier)

if current == goal:

break

for next in grid.neighbors(current):

new\_cost = cost\_so\_far[current] + 1

if next not in cost\_so\_far or new\_cost < cost\_so\_far[next]:

cost\_so\_far[next] = new\_cost

priority = new\_cost + heuristic(goal, next)

heapq.heappush(frontier, (priority, next))

came\_from[next] = current

# Reconstruct path

path = []

curr = goal

while curr != start:

path.append(curr)

curr = came\_from.get(curr)

if curr is None:

return []

path.append(start)

path.reverse()

return path

**📄 4. visualization/logger.py — The Display**

**Think of it like:**

A simple way to **print the grid** and show where the robot is.

python

CopyEdit

def print\_grid(grid, robot\_pos, goal):

for y in range(grid.height):

for x in range(grid.width):

if (x, y) == robot\_pos:

print("R", end=" ")

elif (x, y) == goal:

print("G", end=" ")

elif (x, y) in grid.obstacles:

print("#", end=" ")

else:

print(".", end=" ")

print()

print()

**🏁 5. main.py — The Director**

**Think of it like:**

The script that sets everything up and runs the show.

python

CopyEdit

def run\_simulation():

width, height = 10, 10

obstacles = [(3, y) for y in range(1, 9)] + [(6, y) for y in range(2, 10)]

start = (0, 0)

goal = (9, 9)

grid = Grid(width, height, obstacles)

robot = Robot(grid, start, goal)

path = a\_star\_search(grid, start, goal)

robot.set\_path(path[1:]) # skip current position

print("Initial Grid:")

print\_grid(grid, robot.position, goal)

while robot.position != goal and robot.path:

robot.move\_step()

print\_grid(grid, robot.position, goal)

if \_\_name\_\_ == "\_\_main\_\_":

run\_simulation()

**👀 Sample Output**

A . is open space, # is an obstacle, R is the robot, and G is the goal.

bash

CopyEdit

. . . . . . . . . .

. . . # . . # . . .

. . . # . . # . . .

. . . # . . # . . .

...

As the robot moves, the R changes position.

**🤔 What You Learn From This Project**

* Object-Oriented Programming (classes like Robot, Grid)
* Pathfinding (A\* algorithm)
* Clean architecture and modular code
* Visualization via text console
* GitHub folder structure and workflow