Project Report

on

Route finder using A* Algorithm

Submitted by

GWOS EMMANUEL THIERRY

25MCI10379

Under the guidance of

Mr. Kishan Ahuja

In partial fulfilment for the award of the degree of MASTERS IN ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Chandigarh University

October 2025

Certificate

This is to certify that **Gwos Emmanuel Thierry**, a student of **Master in Artificial Intelligence & Machin Learning (AI & ML)**, has successfully completed the **Minor Project** titled **"Route finder using A* Algorithm"** under the esteemed guidance of **Mr. Kishan Ahuja, Assistant Professor, University Institute of Computing (UIC), Chandigarh University**.

This project was undertaken as a part of the academic curriculum and is submitted in **partial fulfilment of the requirements** for the MAM program. The work presented in this project is a result of **group research, diligent effort, and dedication**, demonstrating the student's ability to apply theoretical knowledge to practical problem-solving.

The project "Route Finder using A* Algorithm" is a web-based application designed to determine the most efficient route between two points using the A (A-Star) pathfinding algorithm. This project combines principles of **Advanced Internet Programming**, **full-stack web development**, and **real-time communication** technologies to deliver a responsive and interactive user experience.

I hereby confirm that this project is an **original work** carried out by the student and has **not been submitted elsewhere** for the award of any other degree, diploma, or certification.

Project Guide:

Mr. Kishan Ahuja

Assistant Professor

University Institute of Computing

Chandigarh University

Acknowledgement

I would like to express my sincere gratitude to **Chandigarh University** and the **University Institute of Computing (UIC)** for providing me with the opportunity to undertake this project, **"Route finder using A* Algorithm"**.

I extend my heartfelt appreciation to my esteemed mentor, **Mr. Kishan Ahuja, Assistant Professor**, for his invaluable guidance, continuous support, and insightful feedback throughout the project. His expertise in **Advanced Programming Language** played a crucial role in the successful completion of this project.

I am also grateful to my friends and peers for their encouragement and discussions, which helped refine my approach. Lastly, I thank my family for their unwavering support and motivation during this project.

This project has been an incredible learning experience, and I hope it serves as a foundation for further exploration in **Advanced Programming Language.**

Gwos Emmanuel Thierry

MCA – (AI & ML)

Chandigarh University

Table of Contents

Abstract

This project **"*Route Finder using A* Algorithm*"** is an intelligent pathfinding application that efficiently determines the shortest route between two points on a grid-based map. The project implements the **A-star (A*) search algorithm**, a widely used and powerful heuristic-based algorithm in Artificial Intelligence. It combines the advantages of **Dijkstra's algorithm** and **Greedy Best-First Search**, using both the actual cost of the path and the estimated cost to the goal to make optimal decisions.

This system allows users to visually create start points, destinations, and obstacles, enabling real-time visualization of how the algorithm explores possible routes and converges on the most efficient path. The A* algorithm calculates the total cost function f(n) = g(n) + h(n), where g(n) represents the actual distance from the start node, and h(n) represents the heuristic estimate to the goal node.

By demonstrating the balance between exploration and exploitation, this project provides insight into how artificial intelligence can solve complex navigation problems. The application has potential use cases in **robotics**, **autonomous vehicles**, and **GPS navigation systems**, showcasing the effectiveness of AI in real-world decision-making and route optimization.

1. Introduction

In today's world of technology and automation, finding the most efficient route from one point to another is an essential problem in various domains such as **navigation systems, robotics, gaming, and logistics**. The *Route Finder using A* Algorithm* is a smart pathfinding system designed to address this problem by identifying the shortest and most optimal path between two locations on a grid.

The *A (A-star) algorithm** is one of the most widely used algorithms in artificial intelligence and computer science for solving pathfinding and graph traversal problems. It is an informed search algorithm that combines the strengths of **Dijkstra's algorithm** (which guarantees the shortest path) and **Greedy Best-First Search** (which focuses on the goal). By using a cost function f(n) = g(n) + h(n), where g(n) is the cost from the start node and h(n) is the heuristic estimate to the goal, A* ensures both efficiency and accuracy.

This project visually demonstrates how A* explores nodes, avoids obstacles, and intelligently selects the optimal route in real time. It allows users to interactively set starting points, destinations, and barriers, offering a clear understanding of how artificial intelligence can be applied to **problem-solving and decision-making**.

The main goal of this project is to provide a practical and educational visualization of how intelligent pathfinding works, showing the balance between **exploration, computation, and optimality** in route planning.

2. Tools and Technologies Used

The development of the *Route Finder using A* Algorithm* project involved several tools, technologies, and libraries that together enabled visualization, user interaction, and efficient algorithm execution. Below are the main tools and technologies used:

1. **Programming Language: Python**
    - ➢ Python was chosen because of its simplicity, readability, and strong community support.
    - ➢ It offers built-in libraries for algorithm implementation, mathematical computation, and graphical interface design.
2. **Library: Pygame**
    - ➢ Pygame is a powerful library used to create graphical applications and 2D games in Python.
    - ➢ In this project, it was used to visualize the grid, colour transitions, and pathfinding process in real time.
    - ➢ It also handles **user input**, such as mouse clicks (to set start, end, and barriers) and keyboard events (to start or reset the pathfinding).
3. **Algorithm: A (A-Star) Pathfinding Algorithm**
    - ➢ The A* algorithm is the core of this project.
    - ➢ It uses **heuristics** and **cost functions** to find the most optimal path from the start node to the goal node.
    - ➢ It combines the benefits of **Dijkstra's Algorithm** (guaranteed shortest path) and **Greedy Best-First Search** (speed and efficiency).
4. **Development Environment: Visual Studio Code**
    - ➢ These IDEs (Integrated Development Environments) were used for coding, debugging, and testing the project efficiently.
    - ➢ They provide syntax highlighting, code completion, and built-in terminal support.

5. **Operating System: Windows**

   ➢ The project is cross-platform and can be run on any OS that supports Python and Pygame.

6. **Other Supporting Libraries:**

   ➢ **Math**: used for heuristic calculations (like Manhattan distance).

   ➢ **PriorityQueue**: manages the open list efficiently during pathfinding.

3. Implementation Steps

The development of the "Route finder using A* Algorithm" was carried out in the following essential steps:

Step 1: Setting Up the Environment

- Installed Python (version 3.x) on the system.
- Installed the Pygame library using the command**: pip install pygame**
- Verified successful installation by running a simple Pygame window program.

Step 2: Designing the Grid Layout

- Defined a grid (2D array) where each cell (called a *node*) represents a possible position on the map.
- Each node can have different states:

  ➢ Start node (Orange) beginning of the route

  ➢ End node (Turquoise) destination point

  ➢ Barrier (Black) obstacle that cannot be crossed

  ➢ Open node (Green) nodes currently being explored

  ➢ Closed node (Red) nodes already explored

  ➢ Path node (Purple) final shortest route found

Step 3: Creating the Node Class

- Implemented a Node class to store:
  - ➢ Its position (row, column)
  - ➢ Colour (state)
  - ➢ Neighbouring nodes (for movement)
  - ➢ Utility methods to check whether a node is a start, end, or barrier.
- The draw () method of each node is used to visually render it on the Pygame window.

Step 4: Implementing the A* Algorithm

- Used two main functions:
  - ➢ g(n): Cost from the start node to the current node.
  - ➢ h(n): Heuristic (estimated cost) from the current node to the goal node, using the Manhattan Distance formula.
  - ➢ f(n): Total cost (f = g + h).
- The algorithm selects the node with the lowest f-score and explores its neighbours until the destination is reached.
- If no path exists, a "No Path Found" message is displayed.

Step 5: Handling User Interaction

- Implemented mouse controls:
  - ➢ Left-click: Set the start, end, or barrier node.
  - ➢ Right-click: Reset any node to its default state.
- Implemented keyboard controls:
  - ➢ SPACE key: Start the pathfinding process.
  - ➢ C key: Clear the grid to start a new search**.**

Step 6: Visualizing the Pathfinding Process

- Used Pygame's drawing functions to dynamically update the grid during the search.

- The process visually shows how the algorithm explores different paths and converges toward the shortest one.

Step 7: Testing and Validation

- Tested with different start and end positions and various obstacle configurations.
- Verified that:
  - ➢ The algorithm finds the shortest path when available.
  - ➢ Displays “No Path Found” when the path is blocked.
  - ➢ The interface responds correctly to user input.

4. Source Code.

```
import pygame
import math
from queue import PriorityQueue

pygame.init()

WIDTH = 600
WIN = pygame.display.set_mode ((WIDTH, WIDTH))
pygame.display.set_caption ("AI Pathfinding Visualizer (A*)")

# Colors
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
GREY = (200, 200, 200)
YELLOW = (255, 255, 0)
PURPLE = (128, 0, 128)
TURQUOISE = (64, 224, 208)
ORANGE = (255, 165, 0)

# Font
FONT = pygame.font.SysFont("arial", 20)

class Node:
```

```
def __init__(self, row, col, width, total_rows):
    self.row = row
    self.col = col
    self.x = row * width
    self.y = col * width
    self.color = WHITE
    self.neighbors = []
    self.width = width
    self.total_rows = total_rows

def get_pos(self):
    return self.row, self.col

def is_closed(self): return self.color == RED
def is_open(self): return self.color == GREEN
def is_barrier(self): return self.color == BLACK
def is_start(self): return self.color == ORANGE
def is_end(self): return self.color == TURQUOISE
def reset(self): self.color = WHITE
def make_start(self): self.color = ORANGE
def make_closed(self): self.color = RED
def make_open(self): self.color = GREEN
def make_barrier(self): self.color = BLACK
def make_end(self): self.color = TURQUOISE
def make_path(self): self.color = PURPLE

def draw(self, win):
```

```
        pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.width))

    def update_neighbors(self, grid):
        self.neighbors = []
            if self.row < self.total_rows - 1 and not grid[self.row + 1][self.col].is_barrier():  # DOWN
            self.neighbors.append(grid[self.row + 1][self.col])
        if self.row > 0 and not grid[self.row - 1][self.col].is_barrier():  # UP
            self.neighbors.append(grid[self.row - 1][self.col])
            if self.col < self.total_rows - 1 and not grid[self.row][self.col + 1].is_barrier():  # RIGHT
            self.neighbors.append(grid[self.row][self.col + 1])
        if self.col > 0 and not grid[self.row][self.col - 1].is_barrier():  # LEFT
            self.neighbors.append(grid[self.row][self.col - 1])

    def __lt__(self, other): return False

def h(p1, p2):
    x1, y1 = p1
    x2, y2 = p2
    return abs(x1 - x2) + abs(y1 - y2)

def algorithm(draw, grid, start, end):
    count = 0
    open_set = PriorityQueue()
    open_set.put((0, count, start))
    came_from = {}
    g_score = {node: float("inf") for row in grid for node in row}
```

```
g_score[start] = 0
f_score = {node: float("inf") for row in grid for node in row}
f_score[start] = h(start.get_pos(), end.get_pos())

open_set_hash = {start}

while not open_set.empty():
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()

    current = open_set.get()[2]
    open_set_hash.remove(current)

    if current == end:
        while current in came_from:
            current = came_from[current]
            current.make_path()
            draw()
        end.make_end()
        start.make_start()
        return True  # Path found!

    for neighbor in current.neighbors:
        temp_g_score = g_score[current] + 1

        if temp_g_score < g_score[neighbor]:
```

```
                came_from[neighbor] = current
                g_score[neighbor] = temp_g_score
                        f_score[neighbor] = temp_g_score + h(neighbor.get_pos(),
end.get_pos())
                if neighbor not in open_set_hash:
                    count += 1
                    open_set.put((f_score[neighbor], count, neighbor))
                    open_set_hash.add(neighbor)
                    neighbor.make_open()

        draw()

        if current != start:
            current.make_closed()

    return False  # No path found

def make_grid(rows, width):
    grid = []
    gap = width // rows
    for i in range(rows):
        grid.append([])
        for j in range(rows):
            node = Node(i, j, gap, rows)
            grid[i].append(node)
    return grid
```

```
def draw_grid(win, rows, width):
    gap = width // rows
    for i in range(rows):
        pygame.draw.line(win, GREY, (0, i * gap), (width, i * gap))
        for j in range(rows):
            pygame.draw.line(win, GREY, (j * gap, 0), (j * gap, width))


def draw(win, grid, rows, width, message=""):
    win.fill(WHITE)
    for row in grid:
        for node in row:
            node.draw(win)
    draw_grid(win, rows, width)
    if message:
        text = FONT.render(message, True, (255, 0, 0))
        win.blit(text, (10, 10))
    pygame.display.update()


def get_clicked_pos(pos, rows, width):
    gap = width // rows
    y, x = pos
    row = y // gap
    col = x // gap
    return row, col


def main(win, width):
    ROWS = 40
```

```
grid = make_grid(ROWS, width)
start = None
end = None
run = True
started = False
message = "Click to set Start, End, and Walls. Press Space to Start."

while run:
    draw(win, grid, ROWS, width, message)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            run = False

        if started:
            continue

        if pygame.mouse.get_pressed()[0]:  # LEFT
            pos = pygame.mouse.get_pos()
            row, col = get_clicked_pos(pos, ROWS, width)
            node = grid[row][col]
            if not start and node != end:
                start = node
                start.make_start()
            elif not end and node != start:
                end = node
                end.make_end()
            elif node != end and node != start:
```

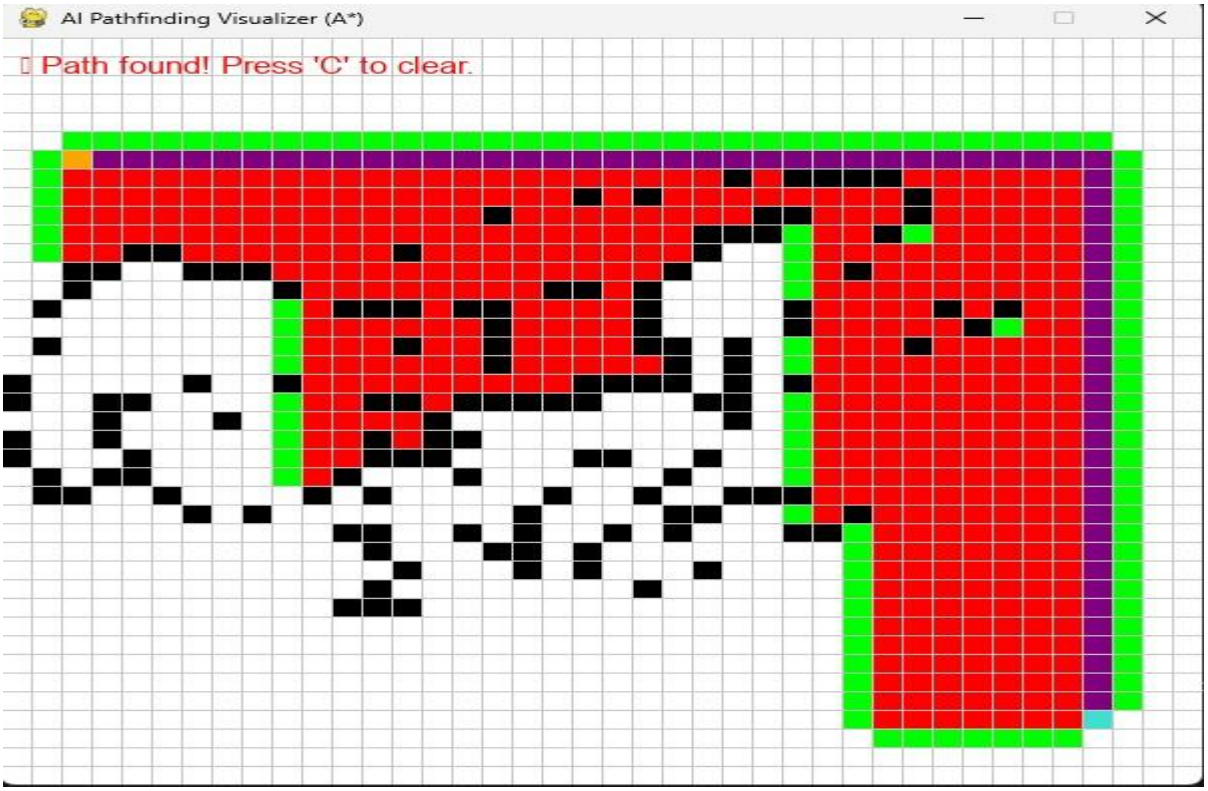
```
                node.make_barrier()

            elif pygame.mouse.get_pressed()[2]:  # RIGHT
                pos = pygame.mouse.get_pos()
                row, col = get_clicked_pos(pos, ROWS, width)
                node = grid[row][col]
                node.reset()
                if node == start:
                    start = None
                elif node == end:
                    end = None

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE and start and end:
                    for row in grid:
                        for node in row:
                            node.update_neighbors(grid)

                     found = algorithm(lambda: draw(win, grid, ROWS, width), grid,
start, end)
                    if not found:
                        message = "❌ No path found! Press 'C' to clear."
                    else:
                        message = "☑ Path found! Press 'C' to clear."

                if event.key == pygame.K_c:
                    start = None
```

```
                end = None
                grid = make_grid(ROWS, width)
               message = "Click to set Start, End, and Walls. Press Space to Start."

    pygame.quit()

main(WIN, WIDTH)
```

5. Screenshots/Result

i. First window.

ii. Second window shortest path found from point A to B



6. Meaning of each colour:

Each colour is represented by its RGB (Red, Green, Blue) values and each colour play a vital role in our program.

- Start point: Orange
- End point: Turquoise
- Path found: Purple
- Barriers: Black
- Visited nodes: Red and Green

7. Future Work

Although the current *Route Finder using A* Algorithm* successfully demonstrates efficient pathfinding on a grid, several enhancements can be made to improve its performance, realism, and user experience in future versions:

- **Integration with Real Maps:** Incorporate APIs such as Google Maps or OpenStreetMap to apply the algorithm on real-world navigation routes instead of a simulated grid.
- **Dynamic Obstacles Handling:** Enable the system to recalculate routes automatically when obstacles appear or disappear (useful in traffic navigation or robotics).
- **Heuristic Optimization:** Experiment with different heuristic functions (e.g., Euclidean, Manhattan, or Diagonal distance) and compare their effects on efficiency and accuracy.
- **3D Pathfinding:** Extend the algorithm to support three-dimensional environments for applications in drones, gaming, or robotics navigation.
- **Multi-Agent Pathfinding:** Implement coordination between multiple agents (e.g., cars, robots) to avoid collisions and plan optimal collective routes.
- **Performance Comparison:** Compare A* with other algorithms such as Dijkstra, BFS, DFS, and Genetic Algorithms to analyse efficiency and scalability.
- **Mobile/Web Application:** Deploy the system as an interactive web or mobile app using Flask, React, or Flutter for real-time visualization and user interaction.

8. Learning Outcomes

During the development of the *Route Finder using A* Algorithm* project, several technical and conceptual skills were learned and strengthened:

- **Algorithmic Understanding:** Gained a deep understanding of informed search algorithms, especially the A* algorithm's balance between optimality and efficiency.
- **Heuristic Design:** Learned how heuristics influence search performance and how to design and test different heuristic functions.

- **Problem Decomposition:** Developed the ability to break down a complex navigation problem into smaller components such as node representation, grid management, and visualization.
- **Programming and Debugging Skills:** Enhanced Python programming proficiency using libraries like **Pygame** for visualization and learned debugging techniques for interactive applications.
- **Visualization Techniques:** Understood how to represent search algorithms visually, showing open and closed nodes, barriers, and final paths in real time.
- **Optimization and Performance Thinking:** Learned to balance accuracy and speed by optimizing loops, conditions, and data structures.
- **Practical Application of AI Concepts:** Understood how theoretical AI algorithms can be applied to solve real-world problems like navigation and route planning.
- **Teamwork and Research Skills:** Improved collaboration, problem-solving, and research abilities when studying algorithmic concepts and implementing them in code.

9. GitHub Link:

https://github.com/gwos23/Route-Finding-using-A-star-algorithm

10. Conclusion

The *Route Finder using A* Algorithm* project successfully demonstrates how artificial intelligence techniques can be applied to solve real-world pathfinding problems efficiently. Through this project, the A* algorithm proved to be both **optimal** and **complete**, meaning it always finds the shortest possible path when one exists, and it can determine when no path is available.

By integrating the algorithm with a graphical interface using **Pygame**, users can **visually interact** with the pathfinding process — setting start and end points, placing obstacles, and observing how the algorithm explores the grid to discover the best route. This not only makes the concept easier to understand but also highlights the power of heuristic-based search methods in AI.

Overall, this project deepened the understanding of **graph traversal**, **heuristic evaluation**, and **real-time visualization** in algorithmic problem-solving. It also lays a foundation for future enhancements, such as implementing diagonal movement, weighted terrain, or integrating real world map data for navigation systems.

To end, the project achieves its primary goal of providing an intuitive and educational demonstration of the *A Pathfinding Algorithm**, combining both **theoretical understanding** and **practical application** in computer science and artificial intelligence.

11. References

1. Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths.* IEEE Transactions on Systems Science and Cybernetics, 4(2), 100–107. (Original paper introducing the A* Algorithm)
2. Russell, S., & Norvig, P. (2020). *Artificial Intelligence: A Modern Approach* (4th Edition). Pearson Education. (Comprehensive reference for search algorithms including A*)
3. GeeksforGeeks. (n.d.). A* Search Algorithm. Retrieved from https://www.geeksforgeeks.org/a-search-algorithm/ (Detailed explanation and pseudocode examples)
4. Red Blob Games. (n.d.). *Introduction to the A* Pathfinding Algorithm.* Retrieved from https://www.redblobgames.com/pathfinding/a-star/introduction.html (Excellent visual and conceptual explanation of A*)
5. Pygame Community. (n.d.). *Pygame Documentation.* Retrieved from https://www.pygame.org/docs/ (Used for graphical visualization of the grid and user interface)
6. Tutorials Point. (n.d.). *Python – Priority Queue.* Retrieved from https://www.tutorialspoint.com/python_data_structure/python_priority_queue.htm (Explains how PriorityQueue is used in implementing A*)
7. GitHub Repository. (n.d.). A* Pathfinding Visualizer Projects. Retrieved from https://github.com/topics/a-star-algorithm (Various open-source implementations for reference)