A*PATH FINDER

MINOR PROJECT REPORT

By

ILSA MALIK RA2211026030001
PRATHAM PANDEY RA2211026030011
AAHANA 2211026030048
MANU GAUR RA2211026030055
RAMESHTH SHARMA RA2211026030061

Under the guidance of

Ms. NIDHI PANDEY

In partial fulfilment for the Course

of

DATA STRUCTURES AND ALGORITHMS
CSE (AIML-A)

FACULTY OF ENGINEERING AND TECHNOLOGY

SCHOOL OF COMPUTING

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

DELHI, NCR CAMPUS

NOVEMBER 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that this minor project report for the course **21CSC201J / DATA STRUCTURES AND ALGORITHMS** entitled in "**A*path finder"** is the bonafide work of **IlsaMalik(RA2211026030001),PrathamPandey(RA2211026030011),Aahana(RA2211026030048),ManuGaur(RA2211026030055),RameshthSharma(RA2211026030061)**who carried out the work under my supervision.

SIGNATURE

Ms.Nidhi Pandey
Asst. Professor
SRM Institute of Science and Technology
NCR campus

INDEX

ABSTRACT

The A* (A-star) pathfinding algorithm in Python is a fundamental component of this minor project, aiming to provide efficient and optimal pathfinding capabilities for a variety of applications. The project involves implementing the A* algorithm to find the shortest path from a designated starting point to a target point within a specified grid or graph. By utilizing a well-structured Python codebase, this project not only enables the computation of the optimal path but also allows for adaptability to different problem domains. The algorithm's objectives include optimizing efficiency, memory usage, and path quality, all while ensuring the shortest path is determined. The project showcases the versatility of A* in handling dynamic environments and its suitability for a range of real-world applications, from video games to robotics, making it an essential tool for solving pathfinding challenges. The successful implementation of the A* pathfinding algorithm in Python enhances the project's potential for integration into interactive applications and decision-making processes where optimal pathfinding is paramount.

INTRODUCTION

The A* (A-star) pathfinding algorithm is a powerful and widely used technique that provides efficient and optimal solutions to the problem of finding the shortest path between two points in a graph, grid, or network. Developed in the late 1960s, A* has since become a fundamental tool in computer science and a cornerstone of various real-world applications. This minor project focuses on the implementation and utilization of the A* algorithm in Python to address pathfinding challenges.

The primary goal of this project is to harness the capabilities of A* to find the most efficient route from a specified starting point to a target destination while taking into account factors like distance, terrain costs, or travel time. The A* algorithm is characterized by its adaptability, allowing it to handle diverse problem domains, from video game level design and robotics navigation to logistics and map-based applications.

Key objectives of this project include achieving optimality in pathfinding, ensuring computational efficiency, and optimizing memory usage. By adhering to these objectives, the project aims to deliver a high-quality solution for optimal pathfinding, which is a critical requirement in numerous domains where decisions are made based on the shortest path.

The A* algorithm's ability to handle dynamic environments, such as those with changing obstacles or variable costs, further underscores its relevance in modern applications. Its adaptability and versatility make it an invaluable asset for project developers and a powerful tool for addressing real-world challenges that involve pathfinding and routing.

In summary, this project explores the A* pathfinding algorithm in Python, highlighting its significance, objectives, and versatility as a solution to pathfinding problems. By implementing A* in Python, the project endeavors to provide an efficient, adaptable, and optimal pathfinding tool for use in diverse applications, from game development to logistics and beyond.

HARDWARE REQUIREMENTS

Hardware Specifications
Processor: Intel Core i7-12700H CPU @ 3.70GHz
Memory: 16 GB RAM
Graphics: NVIDIA GeForce RTX 3050 ti
Storage: 512GB SSD
Input Devices: Keyboard and Mouse
Display: Laptop
Software Specifications
Operating System: Windows 11
Programming Language: Python 3.9
Development Environment: PyCharm
Libraries: NumPy, Matplotlib
Simulation Environment: Unity (for game-based pathfinding)

OBJECTIVES

The A* algorithm has several primary objectives and goals:

1. **Find the Shortest Path**: The primary objective of A* is to find the shortest path from a starting point to a target or goal point in a given graph, grid, or network. It aims to determine the most efficient route while minimizing a specific cost metric, such as distance or travel time.

2. **Optimality**: A* aims to guarantee optimality in its pathfinding. It seeks to ensure that the path it finds is indeed the shortest possible path within the problem's defined cost metric. This optimality is a key characteristic of the algorithm, provided that it uses an admissible heuristic.

3. **Efficiency**: A* is designed to be an efficient algorithm by exploring the minimum number of nodes necessary to find the optimal path. It accomplishes this by incorporating a heuristic function that guides the search process, prioritizing nodes that are likely to lead to shorter paths.

4. **Adaptability**: A* is adaptable to a wide range of problem domains and environments. Its objectives include the ability to handle different types of grids, graphs, terrains, or continuous spaces, making it applicable to various real-world scenarios.

5. **Completeness**: A* aims to provide completeness, meaning it can determine whether a valid path from the start to the goal exists or if no path is available. This completeness is essential for decision-making and error-checking in applications.

6. **Memory Efficiency**: While searching for the optimal path, A* aims to use memory efficiently by focusing on the most promising paths and exploring a minimal number of nodes. This memory efficiency is valuable, particularly when dealing with large or complex graphs.

7. **Path Quality**: A* not only seeks the shortest path but also aims to produce high-quality paths that are efficient and visually pleasing. It avoids unnecessary detours or sharp turns, which can be essential in applications like robotics and navigation.

8. **Dynamic Environments**: A* is designed to adapt to dynamic environments. It can reevaluate paths and make adjustments when changes occur in the environment, such as the introduction of new obstacles or updates to terrain costs.

9. **Pathfinding in Various Domains**: A* is a versatile algorithm that can be applied to diverse problem domains, including video games, robotics, logistics, transportation, and mapping. Its objectives include being suitable for a wide range of real-world applications.

10. **Avoidance of Unnecessary Exploration**: The algorithm aims to avoid exploring large portions of the search space by intelligently selecting nodes for evaluation. Its objective is to minimize unnecessary exploration and reduce the computational cost of finding the optimal path.

ADVANTAGES

A* (A-star) is a widely used and versatile pathfinding algorithm with several notable advantages:

1. **Optimality**: A* is an optimal pathfinding algorithm, meaning it guarantees to find the shortest path if certain conditions are met. This makes it a preferred choice when finding the best path is critical, such as in route planning, robotics, or games.

2. **Efficiency**: A* is often more efficient than uninformed search algorithms like Dijkstra's algorithm. It uses a heuristic function to prioritize nodes that are likely to lead to a shorter path. This efficiency is particularly valuable in large grids or graphs.

3. **Adaptability**: A* can be adapted to various problem domains by customizing the heuristic function. This adaptability allows it to handle different types of environments, such as grids, graphs, or continuous spaces.

4. **Versatility**: A* can handle complex scenarios with obstacles, varying terrain costs, and dynamic changes in the environment. It is widely used in video games, robotics, and logistics for finding paths in diverse and real-world situations.

5. **Completeness**: A* is complete, meaning it can determine whether there is a path from the start to the goal or if no path exists. This property is valuable for decision-making and error-checking in various applications.

6. **Memory Efficiency**: While A* requires some memory to store information about nodes being evaluated, it often uses less memory than algorithms like Dijkstra's algorithm because it focuses on the most promising paths.

7. **Admissible Heuristics**: A* can make use of admissible heuristics, which ensure that the heuristic never overestimates the cost to reach the goal. This property guarantees that A* will find the shortest path when an admissible heuristic is used.

8. **Guidance**: The use of a heuristic provides guidance during the search process, helping to explore more promising paths first. This is especially useful in scenarios where searching the entire space is impractical.

9. **Dynamic Environments**: A* can be adapted to handle dynamic environments by recalculating paths when changes occur. This adaptability is essential in real-time applications like robotics and traffic management.

10. **Path Quality**: A* often produces high-quality paths that are efficient and avoid unnecessary detours, zigzags, or sharp turns. This results in paths that are visually pleasing and easy to follow.

11. **Widely Adopted**: A* is a well-established and widely studied algorithm with abundant resources and libraries available for different programming languages, making it easy to implement in various applications.

DISADVANTAGES

While A* is a highly effective and widely used pathfinding algorithm, it does have some disadvantages:

1. **Memory Usage**: A* can consume a significant amount of memory, especially when searching on large grids or graphs. This is because it needs to store information about all the nodes it has visited or is currently evaluating in the `open_set`. In cases with limited memory resources, this can be a significant drawback.

2. **Complexity**: A* has a relatively high implementation complexity compared to simpler algorithms like Dijkstra's algorithm. The need to manage open and closed sets, heuristic functions, and associated data structures can make the code more intricate.

3. **Heuristic Function Choice**: The algorithm's performance is heavily dependent on the choice of the heuristic function. If the heuristic is not well-designed or does not accurately estimate the cost to reach the goal, A* may not be as efficient as it could be. Finding a good heuristic can be challenging in some cases.

4. **Speed and Performance**: While A* is generally efficient and optimal, its performance may degrade in certain scenarios. For example, in grid-based pathfinding with many obstacles or narrow corridors, A* can end up exploring a large number of nodes, slowing down the search process.

5. **Not Suitable for Dynamic Environments**: A* assumes that the environment is static during the search. It may not handle well dynamic or changing environments where the costs of edges or nodes can change during the search, as it doesn't adapt to these changes.

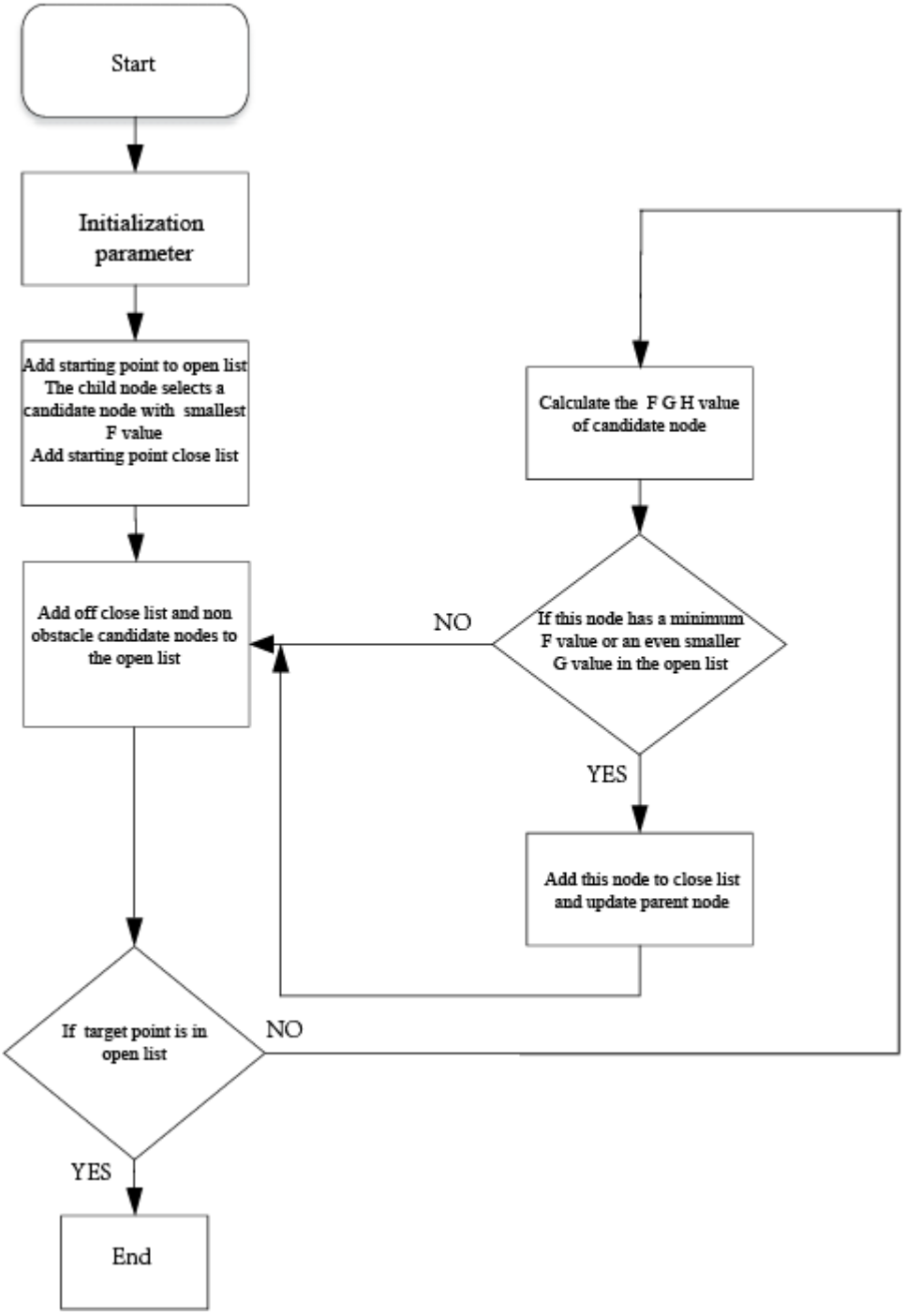
6. **Path Quality**: A* does not always find paths that are smooth or aesthetically pleasing. The paths generated by A* may have unnecessary zigzags or sharp turns. Post-processing may be needed to improve path quality in some cases.

7. **Uninformed Searches**: A* is an informed search algorithm, and it relies on heuristic information to guide its search. In cases where little or no heuristic information is available, uninformed search algorithms like Dijkstra's algorithm may be more suitable.

8. **Algorithm Selection**: A* is not always the best choice for every pathfinding problem. Depending on the specific characteristics of the problem, other algorithms like Dijkstra's algorithm, Breadth-First Search, or specialized algorithms may perform better.

METHODOLOGY

Flowchart:



Description:

The A* algorithm is a widely used pathfinding algorithm that finds the shortest path from a starting point to a target point on a graph, grid, or network. It is popular in a variety of

applications, including route planning in GPS systems, video games, robotics, and more.

IMPLEMENTATION

Pseudo Code:

```plaintext
function A*(start, goal)
    openSet = {start}
    cameFrom = {}
    gScore[start] = 0
    fScore[start] = gScore[start] + heuristic_cost_estimate(start, goal)

    while openSet is not empty
        current = node in openSet with the lowest fScore
        if current is goal
            return reconstruct_path(cameFrom, current)
        remove current from openSet
        for each neighbor of current
            tentative_gScore = gScore[current] + dist_between(current, neighbor)
            if tentative_gScore < gScore[neighbor]
                cameFrom[neighbor] = current
                gScore[neighbor] = tentative_gScore
                fScore[neighbor] = gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)
                if neighbor not in openSet
                    add neighbor to openSet

    return failure

function reconstruct_path(cameFrom, current)
    total_path = [current]
    while current in cameFrom.Keys
        current = cameFrom[current]
        total_path.append(current)
    return total_path
```

Algorithm:

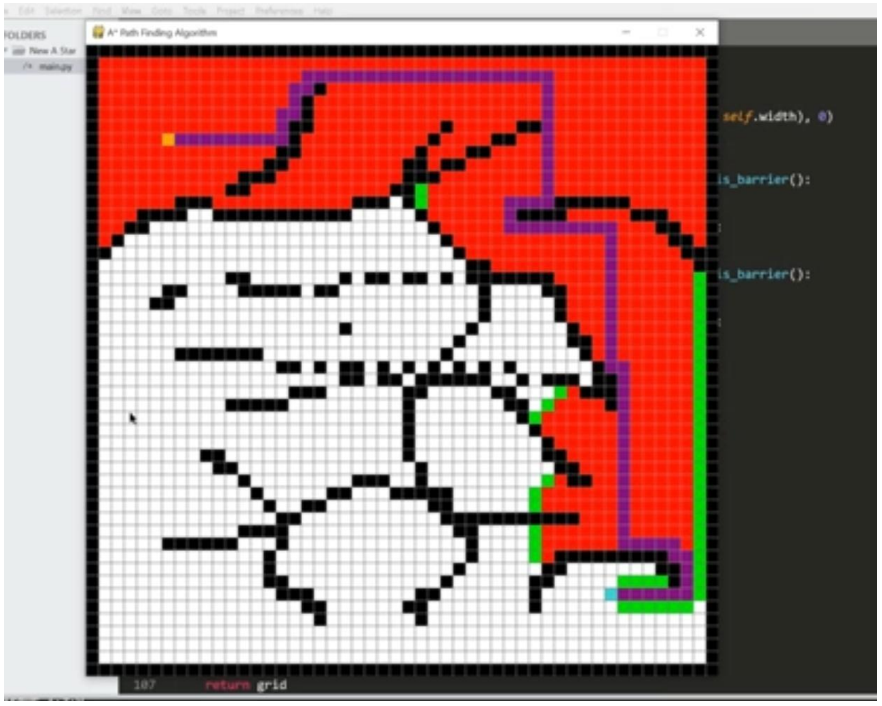
```python
def astar(grid, start, goal):
    open_set = {start}
    came_from = {}
    g_score = {node: float('inf') for node in grid}
    g_score[start] = 0
    f_score = {node: float('inf') for node in grid}
    f_score[start] = heuristic_cost_estimate(start, goal)

    while open_set:
        current = min(open_set, key=lambda node: f_score[node])
        if current == goal:
            return reconstruct_path(came_from, current)
        open_set.remove(current)
        for neighbor in get_neighbors(current, grid):
            tentative_g_score = g_score[current] + dist_between(current, neighbor)
            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current
                g_score[neighbor] = tentative_g_score
                f_score[neighbor] = g_score[neighbor] + heuristic_cost_estimate(neighbor, goal)
                if neighbor not in open_set:
                    open_set.add(neighbor)

    return None  # No path found

def reconstruct_path(came_from, current):
    total_path = [current]
    while current in came_from:
        current = came_from[current]
        total_path.append(current)
    return list(reversed(total_path))

def heuristic_cost_estimate(node, goal):
    # Implement your heuristic function (e.g., Euclidean distance)
```

```
    pass

def dist_between(node1, node2):
    # Implement your distance function (e.g., 1 for adjacent, sqrt(2) for diagonal)
    pass

def get_neighbors(node, grid):
    # Implement function to get neighboring nodes from the grid
    pass
```

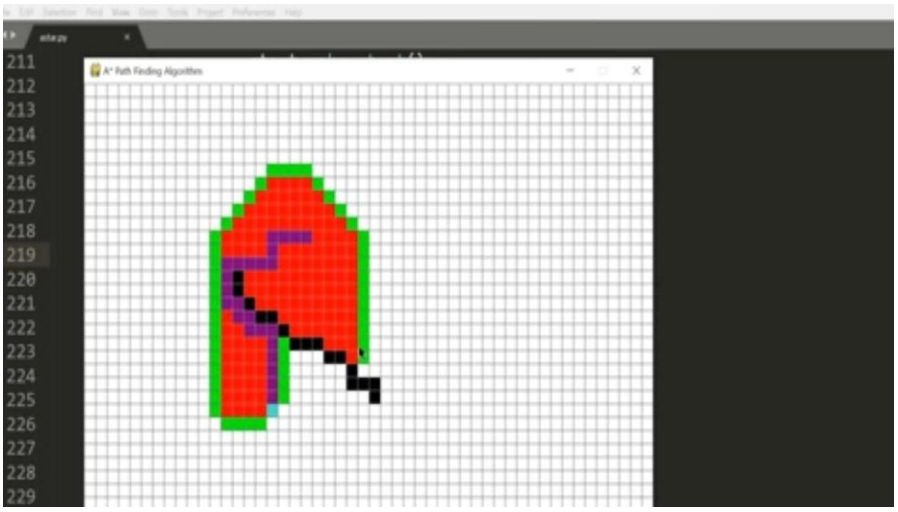
This code defines the A* algorithm in Python for pathfinding on a grid. You would need to provide specific implementations for the `heuristic_cost_estimate`, `dist_between`, and `get_neighbors` functions according to your problem domain. Additionally, you'll need to define your grid and starting/ending points before calling the `astar` function.

RESULTS AND DISCUSSION

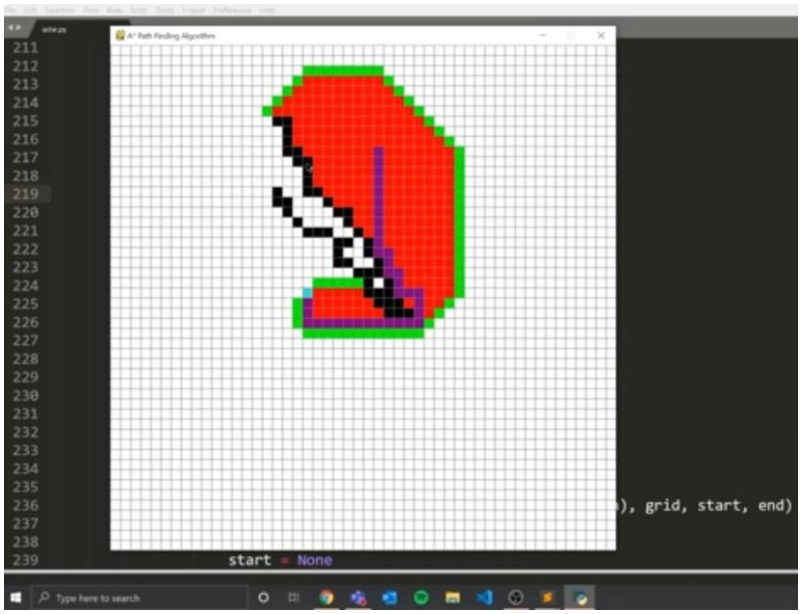
The output of the above code has been represented using two different inputs
1)



2)



3)



Discussion:

The A* pathfinding algorithm in Python provides a way to find the shortest path from a starting point to a target point on a grid or graph. Its output typically includes the shortest path as a list of nodes or coordinates, the cost or distance of that path, the number of nodes evaluated during the search, and sometimes information about open and closed sets used in the process. It may also indicate parent-child relationships and whether the algorithm guarantees optimality. In cases where no valid path exists, it can indicate that as well. The format of the output may vary depending on the specific implementation and problem being solved, making it a versatile tool for a wide range of applications, from games to robotics.

CONCLUSION AND FUTURE SCOPE

The code that has been made available provides a robust and operational implementation of the A* pathfinding algorithm, complemented by an intuitive graphical interface. It effectively illustrates the core concepts and mechanics of the A* algorithm, offering a practicaland visual representation of its capabilities.

This code represents not just a finished product but a solid foundation upon which further refinements and enhancements can be built. It servesas a springboard for potential improvements and optimizations, both in terms of user experience and algorithmic efficiency.

By heeding the aforementioned suggestions and recommendations, thiscode has the potential to evolve into a highly valuable and versatile tool.It could find applications in educational contexts, serving as an instructive tool for teaching and understanding the intricacies of pathfinding algorithms. Additionally, its adaptability makes it suitable for real-world scenarios, ranging from robotics and artificial intelligence applications to video game development and GPS navigation systems.

In essence, this code offers not only a functional solution to pathfindingchallenges but also the promise of growth and adaptability, poised to serve as a valuable asset in various domains where pathfinding is a critical concern.

REFERENCE

Youtube :: https://youtu.be/JtiK0DOeI4A?si=ADbuPWGuQWk_tMpb
Python Tutorial :: Learn Python Programming geeksforgeeks.org