RoamIO An Indoor Navigation Robot

Group Number: 6
CSE440 Final Project Report
Section: 1

1.Naima Siddika Toha
ID: 2013506642

2.Md Saidur Rahman Antu
ID: 1911512042

3. Saima Arafin Smrity
ID: 1913061042

4. Syed Sakibul Haque Sami
ID: 1620298042

Abstract**—** This project presents the design and implementation of an autonomous robot navigation system in an environment. The system gives permission a robot to move to the destination which is selected by the user. During this time, robot will avoid obstacles. The robot uses the A* search algorithm, which provides a way of finding the shortest path in a grid- based environment. The robot's movement is controlled by physics- based motion. The path is visualized using line rendering. In order to improve user experience, a camera controller is added that automatically switches between a top-down and a follow- camera view. After testing in the environment, the robot shows that it can reach successfully to the main destination while avoiding obstacles or blocked areas. This project demonstrates how A* algorithms, obstacle detection while movement, and visualization techniques can be combined to create a robot navigation system.

Keywords**— A* algorithm, autonomous robot navigation, Unity, C#, AI.**

I. INTRODUCTION

In today's world, autonomous navigation is challenging in both computer science and robotics. For a robot to be useful, it needs to reach in the main destination properly by avoiding obstacles or blocked things on the way. In real- world systems, this is the most difficult task because it requires complex algorithms, continuous sensor feedback, and real- time adjustment. The same concepts can be explored in the simulation environments, where the focus is on building the logic rather than dealing with the limitations of hardware.

In this project, we built a robot simulation that can move toward any location which is chosen by the user. The user simply clicks on the ground and the system starts to calculate a safe path by using A* pathfinding algorithm. The robot follows this path step by step and makes sure while moving it avoids obstacles in the environment. Since the environment of the robot is grid- based, so it's each part of the space is divided into small cells that are wither blocked with obstacles or walkable. The A* algorithm uses this grid to decide which route is the shortest and safest for moving the robot. A* algorithm helps to find the path in the shortest way. The robot uses A* search algorithm because it finds the shortest path or moving path between a starting node and destination or goal node.

To make the simulation more realistic and engaging, a camera system was also included. When the robot is idle, the camera stays in a top-down position so that the user of the robot can clearly investigate and can see the whole environment properly. When the robot starts moving, the camera automatically switches to follow view and make the simulation more dynamic.

The project carries together ideas from Artificial Intelligence (AI), robotics, game development, and computer science. It shows how a classic algorithm like A* can be used to solve navigation problems in a simple but effective way. Though it is a simulation, the concepts applied here are the same as those used in real- world applications such as house cleaner robot, delivery robot, self-driving cars, and drones.

II. LITERATURES REVIEW

A. Formal Basis for the Heuristic Determination of Minimum Cost Paths

Many algorithms have proposed to help agents move efficiently using an environment. The earliest foundation came from Hart, Nilsson, and Raphael (1968), who introduced the A* algorithm. They showed how heuristic value works. This heuristic refers an estimate of how far the goal is. They showed in the A* algorithm, heuristic value uses in regular search process which makes the process both fast and easier. In Dijkstra's algorithm, it explores almost everything but A* focuses only on the most promising destination. This process and correctness is the main reason it has become so widely used in robotics, and game.

B. Anytime Dynamic A: An anytime, replanning algorithm

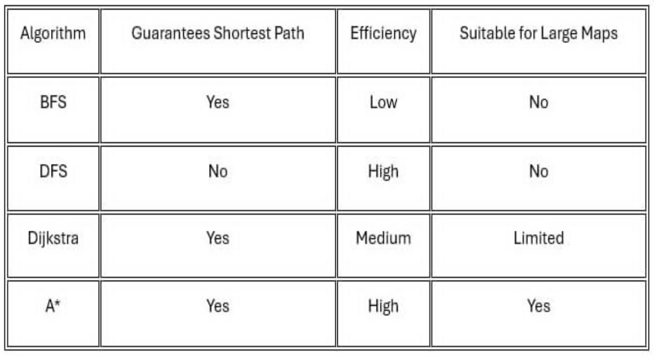
Another improvement came from Likhachev et al. (2005), who created Anytime Dynamic A*. This algorithm finds a quick but most probably non-optimal path. Then improves it step by step as time allows. If environment start to change , it updates the solution without starting over again. This tell us how A* can be extended for situations where both speed and adaptability are necessary.

III. METHODOLOGY

The system was designed as a combination of six Unity scripts which work together. The CameraController script manages how the camera will work. When the robot is not moving it stays in a top- down position and when the robot starts to move the camera switch to following view. The ClickToSetDestination script controls mouse input. When the user clicks on the floor, the click location is processed to detect where the robot should go or should not go. The grid is updated to investigate and see the obstacles or blocked areas. Then the pathfinding algorithm like A* is used to calculate the path. The GridManager script creates a virtual grid over the environment. Each and every position represent as a node. These nodes can either blocked by an obstacle or can be walkable. The node class stores information for each grid position, including it is walkable and its pathfinding costs.
The PathFinding script is the main part of this project where the A* algorithm is implemented. It calculates the shortest path from the robot's current position to the main destination by using heuristic value. The algorithm uses three values for each node. gCost for the distance from the start, hCost for the estimated didtance, to the target, fCost for which is the sum of the two. The algorithm quickly finds the best route by expanding the node with the lowest fCost.
The final script, the RobotController, which helps to control the robot's motion. The path is considered as a series of waypoints. The robot moves from one waypoint to the next until the main destination is reached. In this work, the movement uses Unity's physics system to make it realistic project work. A line renderer is used to draw the planned path above the ground.

.

TABLE I
COMPARISON BETWEEN FOUR ALGORITHMS



Algorithm	Guarantees Shortest Path	Efficiency	Suitable for Large Maps
BFS	Yes	Low	No
DFS	No	High	No
Dijkstra	Yes	Medium	Limited
A*	Yes	High	Yes

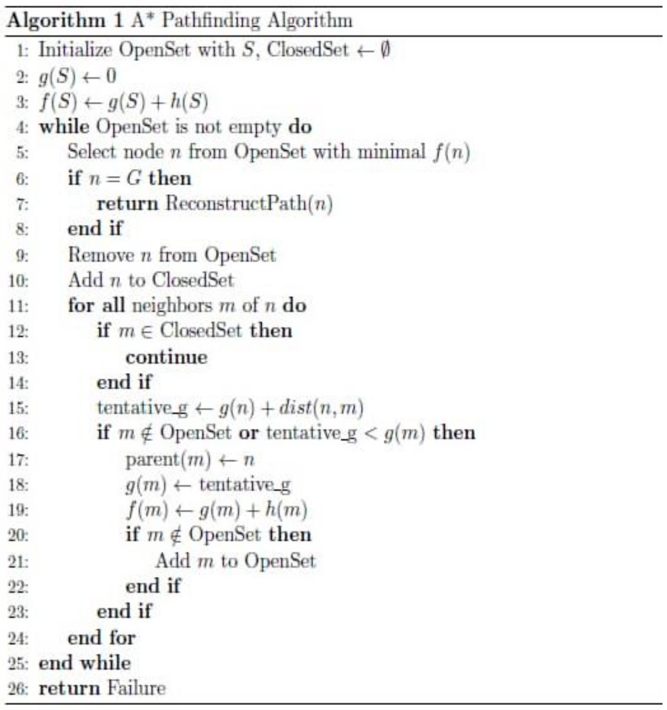
A. Equations

The A* evaluation function defined as:

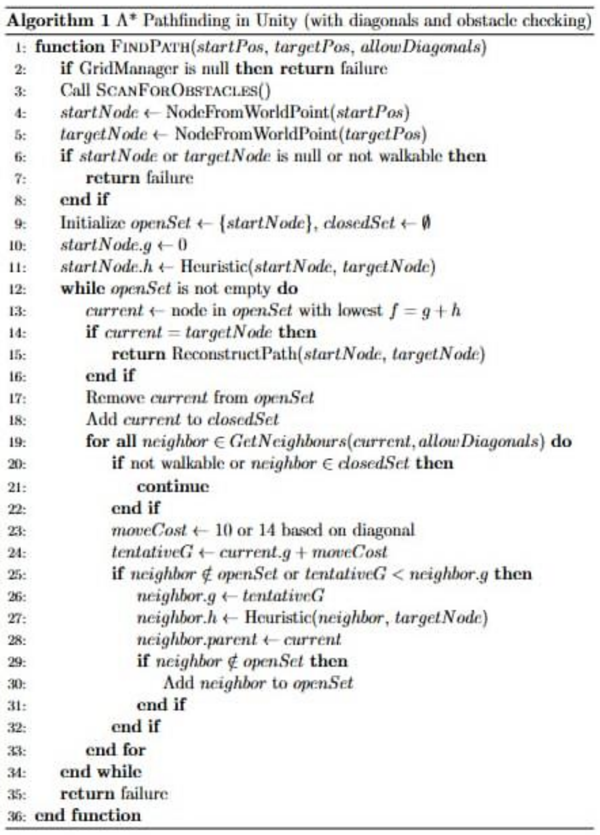
$$F(n)= g(n) + h(n) \tag{1}$$

Where, f(n)= estimated total cost of the cheapest solution through node n.
g(n): cost from the start node to n
h(n)= heuristic estimate of the cost from n to the goal.

B. Algorithms



```
Algorithm 1 A* Pathfinding Algorithm
Initialize OpenSet with $S$, ClosedSet $\leftarrow \emptyset$
$g(S) \leftarrow 0$
$f(S) \leftarrow g(S) + h(S)$
while OpenSet is not empty do
    Select node $n$ from OpenSet with minimal $f(n)$
    if $n = G$ then
        return ReconstructPath($n$)
    end if
    Remove $n$ from OpenSet
    Add $n$ to ClosedSet
    for all neighbors $m$ of $n$ do
        if $m \in$ ClosedSet then
            continue
        end if
        tentative_g $\leftarrow g(n) + dist(n, m)$
        if $m \notin$ OpenSet or tentative_g $< g(m)$ then
            parent($m$) $\leftarrow n$
            $g(m) \leftarrow$ tentative_g
            $f(m) \leftarrow g(m) + h(m)$
            if $m \notin$ OpenSet then
                Add $m$ to OpenSet
            end if
        end if
    end for
end while
return Failure
```



```
Algorithm 1 A* Pathfinding in Unity (with diagonals and obstacle checking)
function FINDPATH($startPos$, $targetPos$, $allowDiagonals$)
    if GridManager is null then return failure
    Call SCANFOROBSTACLES()
    $startNode \leftarrow$ NodeFromWorldPoint($startPos$)
    $targetNode \leftarrow$ NodeFromWorldPoint($targetPos$)
    if $startNode$ or $targetNode$ is null or not walkable then
        return failure
    end if
    Initialize $openSet \leftarrow \{startNode\}$, $closedSet \leftarrow \emptyset$
    $startNode.g \leftarrow 0$
    $startNode.h \leftarrow$ Heuristic($startNode$, $targetNode$)
    while $openSet$ is not empty do
        $current \leftarrow$ node in $openSet$ with lowest $f = g + h$
        if $current = targetNode$ then
            return ReconstructPath($startNode$, $targetNode$)
        end if
        Remove $current$ from $openSet$
        Add $current$ to $closedSet$
        for all $neighbor \in GetNeighbours(current, allowDiagonals)$ do
            if not walkable or $neighbor \in closedSet$ then
                continue
            end if
            $moveCost \leftarrow$ 10 or 14 based on diagonal
            $tentativeG \leftarrow current.g + moveCost$
            if $neighbor \notin openSet$ or $tentativeG < neighbor.g$ then
                $neighbor.g \leftarrow tentativeG$
                $neighbor.h \leftarrow$ Heuristic($neighbor$, $targetNode$)
                $neighbor.parent \leftarrow current$
                if $neighbor \notin openSet$ then
                    Add $neighbor$ to $openSet$
                end if
            end if
        end for
    end while
    return failure
end function
```

IV. IMPLEMENTATION

A. Programming Language

The project was built by using C# programming language.

B. Robot Setup

A rigidbody was included to the robot to allow a physics- based movement. Movement was managed step by step toward waypoints.

C. Grid and Path Visualization

A line renderer was figured to display paths in green colour. The floor was divided into a grid using the MeshRenderer and MeshFilter.

D. Obstacle Detection

By using the Rigidbody, the robot moved step by step to each waypoint. When it goes to the close to a waypoint , it moves to the next until it reaches to the target. These created nodes for corresponding nodes were marked as unwalkable.

E. User Input and Target Selection

Unity's new Input System was used for detection of clicks. From the mouse position a raycast determined the point on the floor that the user clicked.

F. Pathfinding and Navigation

A pathfinding algorithm calculated a path from the robot's current position to the main destination and avoiding unwalkable nodes. After reaching the destination, the line Renderer was cleared to remove the path which was displayed.

V. DISCUSSION

The project demonstrated the effectiveness work of the A* algorithm in a grid- based environment. We also used C# programming language to do this work. The robot was able to avoid obstacles without need any kind of advanced sensors. We were able to apply our AI knowledge here. The integration of multiple scripts worked perfectly. However, some challenges were observed while doing the work. The path finding performance decreases if the grid becomes very large because more nodes must be checked. The robot's movement is limited from straight lines to waypoints. Obstacles were detected using simple sphere checks. As a result, it may not manage complex shapes.

VI. Conclusion

.Our project successfully implemented robot navigation system using the A* algorithm. The robot can move toward main destination selected by users. While working the robot can detect obstacles and avoid those to go to the main path. A* algorithm helps the robot to go the destination within the shortest path. While there are few limitations, the project demonstrate the core principles of autonomous navigation in a simulated environment successfully. It also foundation for future work on more advanced robotics, game development, and AI navigation techniques.

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. https://doi.org/10.1109/TSSC.1968.300136

[2] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2005). *Anytime Dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 15, No. 1, pp. 262–271). AAAI Press. https://doi.org/10.1609/icaps.v15i1.13068

[3] Unity Technologies. (2023). *Unity User Manual: Rigidbody component*. Unity. https://docs.unity3d.com/Manual/class-Rigidbody.html