MAZE SOLVER

USING A* ALGORITHM

MINI PROJECT REPORT

Submitted by

GOKULANAND.P(202209013)

ARUNKUMAR.K(202209006)

KANAGA BALA.R(202209023)

in

19AD452 –ARTIFICIAL INTELLIGENCE LABORATORY

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

MEPCO SCHLENK ENGINEERING COLLEGE

SIVAKASI

MAY 2024

MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI

AUTONOMOUS

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

BONAFIDE CERTIFICATE

This is to certify that it is the bonafide work of **"GOKULANAND P(202209013),ARUNKUMAR.K(202209006),KANAGABALA.R(202209023)"**for the mini project titled **"MAZE SOLVER USING A*SEARCH"** in 19AD452 – Artificial Intelligence Laboratory during the fourth semester June 2024 – November 2024 under my supervision.

SIGNATURE

Dr.A.Shenbagarajan,
Associate Professor,
AI&DS Department,
Mepco Schlenk Engg., College,
Sivakasi - 626 005.

SIGNATURE

Dr. J. Angela Jennifa Sujana,
Professor & Head,
AI&DS Department
Mepco Schlenk Engg., College,
Sivakasi - 626 005

ACKNOWLEDGEMENT

I sincerely thank our respected principal **Dr.S.Arivazhagan** principal for providing necessary facilities to carry out this work successfully.

I wish to express my sincere gratitude to **Dr.J.Angela Jennifa Sujana** BE, MTech,PhD Professor and Head of the Artificial Intelligence & Data Science department for her stimulating support and encouragement for the completion of this project work.

I am grateful to our Project guides and **Dr.A.Shenbagarajan**Associate Professor BE,ME,PhD and **Dr.S.Shiny** Assistant Professor (Sr.Grade) BE,ME, PhD.Department of Artificial intelligence & DataScience for their insightful comments and valuable suggestions which helped me to complete this project work successfully.

My sincere thanks to our revered **faculty members** and **lab technicians** for their help over this project work.

ABSTRACT

The A* search algorithm is a widely used and powerful method for solving maze navigation problems due to its efficiency and effectiveness in finding the shortest path. This algorithm combines the strengths of Dijkstra's algorithm and the heuristic approach of best-first search, enabling it to intelligently traverse the maze by evaluating the cost of paths. A* search employs a priority queue to explore nodes, utilizing a cost function that sums the path cost from the start node to the current node and an admissible heuristic estimate of the cost from the current node to the goal. This ensures that the algorithm not only explores the most promising paths first but also guarantees optimality and completeness. Through iterative expansion of the least-cost node, A* effectively navigates through complex maze structures, dynamically adjusting its path based on real-time evaluations. This adaptability makes A* particularly suited for various applications, from robotics and video game development to geographic mapping and artificial intelligence research.

TABLE OF CONTENTS

CHAPTER-1

1.INTRODUCTION

1 .1 PROJECT DESCRIBTION:

The Maze Solver project aims to implement an interactive maze-solving algorithm using the A* search algorithm in Python, leveraging the Pygame library for visualization. The project will create a graphical interface where users can generate mazes, set start and goal points, and observe the A* algorithm in action as it finds the shortest path through the maze. This project will not only demonstrate the efficiency and effectiveness of the A* algorithm but also provide a hands-on tool for visualizing pathfinding algorithms.

1.1.1KEY FEATURES:

- **Maze Generation**

  Users can generate random mazes or manually design their own using an intuitive graphical interface.

- **Interactive Interface**

  The Pygame interface will allow users to place start and goal points within the maze.

- **A Search Algorithm***

  The core of the project is the implementation of the A* search algorithm, which will be used to find and display the shortest path from the start to the goal.

- **Visualization**

  As the A* algorithm progresses, the interface will visually update to show the nodes being explored, the current path being considered, and the final shortest path.

- **User Control**

  Users will have controls to start, pause, and reset the maze-solving process, allowing for step-by-step observation of the algorithm.

1.2 Project Objectives:

1.2.1.Maze solver using A*Algorithm:

Provide a foundational understanding of pathfinding algorithms, with a focus on the A* search algorithm.Explain the importance of pathfinding in various fields such as robotics, video games, and navigation systems.Discuss the theory behind different pathfinding algorithms, highlighting the

advantages and limitations of each.

1.2.2.Implementation of the A* Search Algorithm:

- **Objective**

  Implement the A* search algorithm in Python to solve a maze.

- **Details**

  Develop a Python module (astar.py) that encapsulates the A* algorithm.Ensure the algorithm efficiently finds the shortest path by balancing the exploration of nodes using the cost function f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n).Implement a heuristic function that provides admissible estimates to guide the search.

1.2.3.Maze Generation and Representation:

- **Objective**

  Create a system to generate and represent mazes in a grid format.

- **Details**

  Develop a Maze class in maze.py to create and manage the maze structure.Include functionality for both random maze generation and manual design.Represent the maze using a 2D grid, with open spaces and walls.

1.2.4. Graphical User Interface with Pygame:

- **Objective**

  Build an interactive graphical interface using Pygame to visualize the maze and the A* algorithm's pathfinding process.

- **Details**

  Create a visualization.py module to handle the graphical display and user interactions.Allow

users to draw mazes, set start and goal points, and initiate the pathfinding process.Visualize the algorithm's progress, showing explored nodes, the current path under consideration, and the final shortest path.

1.2.5. User Interaction and Control:

- **Objective**

Provide a user-friendly interface that allows users to control the maze-solving process.

- **Details**

Implement mouse interactions for users to place start and goal points.Provide controls to start, pause, and reset the algorithm.Display real-time updates of the algorithm's progress, enhancing understanding of the pathfinding process.

1.2.6. Educational and Practical Insights:

- **Objective**

Offer educational value and practical insights into the functioning of the A* search algorithm and its applications.

- **Details**

Ensure the project is well-documented, with comments and explanations for each part of the code.Create a comprehensive guide or tutorial that explains how the algorithm works, how to run the project, and how to interpret the visualizations.Discuss potential extensions or variations of the project, such as implementing different heuristics or exploring other pathfinding algorithms.

CHAPTER-2

2.ARCHITECTURE

2.1 Block Diagram:

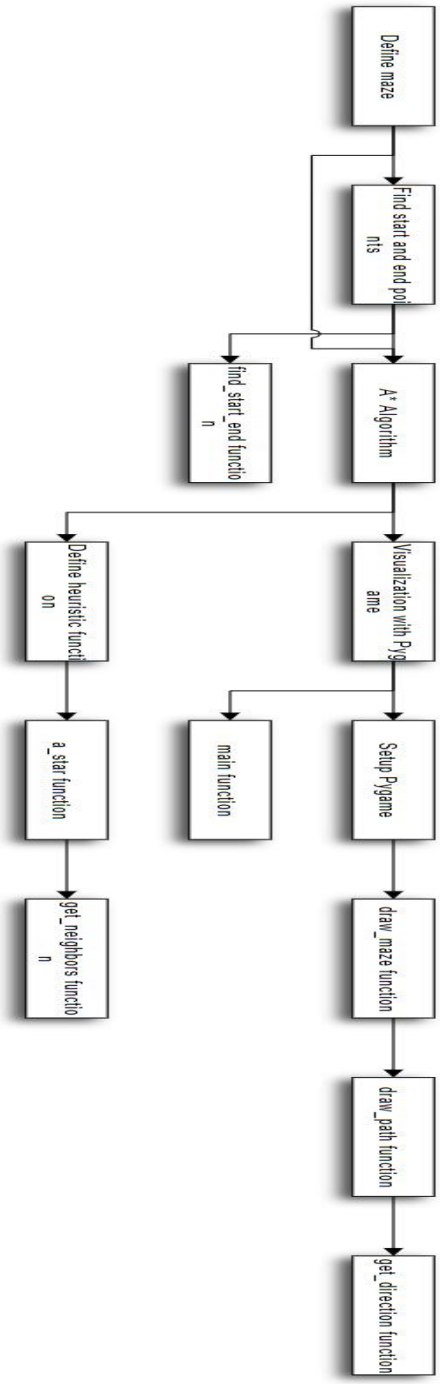

Figure No: 2.1.BlockDiagram

TOOL USED:

Heapq:

The heapq module in Python provides an efficient way to implement the heap queue algorithm, also known as the priority queue algorithm. A heap is a specialized binary tree-based data structure that satisfies the heap property. In a min-heap, the smallest element is always at the root, whereas in a max-heap, the largest element is at the root. Python's heapq module provides functions to create and manipulate heaps in an efficient manner using a regular list as the underlying data structure. The primary feature of a heap is that it allows for the fastest retrieval of the smallest (or largest) element, which is useful in various applications like priority queues, scheduling, and graph algorithms.

Random:

The random module in Python is a powerful and versatile tool for generating pseudo-random numbers and performing random operations. This module is widely used in various applications such as simulations, games, testing, and security to introduce an element of randomness. The random module provides a range of functions to generate random numbers, choose random elements, shuffle sequences, and produce random values with specific distributions.The core of the random module is the random() function, which generates a random float number between 0.0 and 1.0. This function forms the basis for many other functions in the module. For example, to generate a random float within a specified range, you can use the uniform(a, b) function, which returns a random float between a and b. Similarly, the randint(a, b) function returns a random integer between a and b, inclusive. If you need a random integer from a specified range, the randrange(start, stop, step) function allows you to generate an integer within the given range and step.

Pygame Library:

A set of Python modules designed for writing video games, but also useful for any graphical applications. It provides functionalities for rendering graphics, handling events, and updating the display.Pygame is a cross-platform set of Python modules designed for creating video games and multimedia applications. It is built on top of the SDL library, providing an easy-to-use interface for creating graphical applications.

2.3.Key Components of Pygame:

2.3.1.Initialization:

pygame.init()

The line pygame.init() is a crucial step in initializing the Pygame library, which is widely used for developing multimedia applications such as games in Python. Pygame is built on top of the SDL (Simple DirectMedia Layer) library and provides functionalities for handling graphics, sound, and input devices like the keyboard and mouse. When you call pygame.init(), you are preparing the Pygame modules for use in your application by initializing all the necessary components.To understand the importance of pygame.init(), it's essential to delve into what it does under the hood. This function initializes all the imported Pygame modules that require initialization. Pygame consists of various modules such as display, mixer, font, and joystick, each responsible for different aspects of multimedia handling. When you call pygame.init(), it iterates through these modules and initializes each one, making them ready to be used in your program.One of the primary tasks of pygame.init() is to initialize the display module. This module is responsible for creating a window or screen where the graphical content of your game or application will be rendered. Without initializing the display module, you would not be able to create or manipulate windows and surfaces, which are fundamental to any graphical application.

2.3.2.Display:

pygame.display.set_mode((width, height))

The function pygame.display.set_mode((width, height)) is used to create a new window or screen for a Pygame application. By specifying the dimensions as a tuple, such as (800, 600), you define the width and height of the window in pixels. This function initializes the display module, setting up the window where all graphics will be rendered. The function returns a Surface object, which represents the screen. This Surface is the primary canvas where you will draw all your game elements, including shapes, images, and text. The display surface is essentially the main drawing area for your game, and you can perform various drawing operations on it. Additional parameters can be passed to pygame.display.set_mode to control the display mode, such as enabling fullscreen or hardware acceleration, which can enhance performance or provide a different user experience.

pygame.display.flip()

Pygame.display.flip() is a function that updates the entire display surface to the screen. Pygame uses a technique called double buffering to prevent flickering and tearing of the graphics. When you draw on the display surface, you are actually drawing on an off-screen buffer. The pygame.display.flip() function swaps this off-screen buffer with the on-screen buffer, making all your drawing operations visible. This process ensures that all changes made since the last flip are rendered on the screen. By using pygame.display.flip(), you update the entire screen at once, which is critical for maintaining smooth and coherent graphics in your application.

2.3.3.Surfaces:

pygame.Surface: Represents images or portions of the screen. All graphics in Pygame are drawn on surfaces.

pygame.Surface.fill(color)

The method pygame.Surface.fill(color) fills the entire surface with a specified color. This is a straightforward way to clear the screen or set a background color. The color parameter is typically a tuple representing an RGB color, such as (255, 0, 0) for red. When you call this method on the display surface, it fills the entire surface with the chosen color, effectively erasing any previous drawings. This method is commonly used at the beginning of each frame to clear the screen before drawing new content, ensuring that there are no remnants of previous frames.

2.3.4.Drawing Functions:

pygame.draw.rect(surface, color, rect, width=0)

The pygame.draw.rect(surface, color, rect, width=0) function is used to draw rectangles on Pygame surfaces. It accepts parameters specifying the surface to draw on, the color of the rectangle, the position and size of the rectangle, and an optional parameter to control the width of the rectangle's border. This function is commonly used to draw shapes or backgrounds in Pygame applications.

pygame.draw.line(surface, color, start_pos, end_pos, width=1)

The pygame.draw.line(surface, color, start_pos, end_pos, width=1) function is employed to draw lines on Pygame surfaces. It takes arguments indicating the surface to draw on, the color of the line, the starting and ending positions of the line, and an optional parameter to determine the line's width. This function is useful for drawing lines and connecting points in Pygame graphics.

2.3.5.Event Handling:

pygame.event.get()

The pygame.event.get() function retrieves events from the Pygame event queue, which contains user input events such as key presses, mouse movements, and window events. By calling this function, you can access a list of events currently in the queue and handle them accordingly in your Pygame application. This function is essential for implementing interactive features and responding to user actions within a Pygame program.

pygame.event.type()

The pygame.event.type attribute is used to access the type of an event in Pygame. Each event in Pygame has a type associated with it, such as QUIT for window close events, KEYDOWN for key press events, and MOUSEBUTTONDOWN for mouse click events. By accessing the type attribute of a Pygame event, you can determine the type of event that has occurred and respond accordingly in your application. This attribute is crucial for event handling in Pygame, allowing you to differentiate between different types of user input and system events.

2.3.6.Clock:

pygame.time.Clock()

The pygame.time.Clock() function creates a Clock object in Pygame, which is used to control the timing of your game or application. This Clock object provides methods for regulating the frame rate and measuring elapsed time, ensuring that your application runs smoothly and consistently across different systems. By creating a Clock object, you can manage the frame rate of your game loop and synchronize game updates with real-time performance. This function is essential for

maintaining a stable and predictable frame rate in Pygame applications, which is crucial for delivering a smooth and enjoyable user experience.

clock.tick(fps)

Ensures The clock.tick(fps) method is used to control the frame rate of your Pygame application. By calling this method within your game loop and passing the desired frames per second (fps) as an argument, you can regulate the rate at which your game updates and renders frames. This method calculates the time elapsed since the last call and delays execution to achieve the specified frame rate. By adjusting the fps parameter, you can optimize performance and balance between smooth animation and computational efficiency. This method is essential for managing the frame rate of your Pygame application and ensuring consistent performance across different hardware platforms.

2.4 IMPLEMENTATION STEPS:

2.4.1.Setup:

- **Objective**

  Establish the basic structure of the project.

- **Details:**

  Create a project directory.Set up subdirectories and files for different components like the maze generator, A* algorithm implementation, and visualization.Ensure Python and Pygame are installed.

2.4.2. Design the Maze Structure:

- **Objective**

  Define how the maze will be represented in the program.

- **Details**

Decide on a grid-based representation for the maze, where each cell can either be a wall or an open space.Implement a class to handle the maze, including methods for initializing the grid and creating walls.

2.4.3. Maze Generation:

- **Objective**

Create a method to generate the maze.

- **Details**

Implement methods to generate a random maze by randomly placing walls.Ensure the maze generation method creates a solvable maze with a start and goal point.

2.4.4. A* Algorithm Implementation:

- **Objective**

Implement the A* search algorithm for pathfinding.

- **Details**

Define the A* algorithm's data structures, including open and closed lists, cost functions (g and h scores), and the heuristic function.Implement the main loop of the A* algorithm to explore nodes, update scores, and find the shortest path from start to goal.

2.4.5. Path Reconstruction:

- **Objective**

After reaching the goal, reconstruct the path taken.

- **Details**

Implement a method to backtrack from the goal node to the start node using a came_from map, building the path.

2.4.6. Neighbor Nodes:

- **Objective**

  Define how to get the neighboring nodes for a given node.

- **Details**

  Ensure the method correctly identifies valid neighbors (i.e., not walls and within grid boundaries).Consider the movement costs for different directions (e.g., straight vs. diagonal).

2.4.7. Setting Up Pygame Visualization:

- **Objective**

  Initialize the Pygame library and set up the display window.

- **Details**

  Create a display window of appropriate size based on the maze dimensions.Set up a clock to control the frame rate of the visualization.

2.4.8. Drawing the Maze:

- **Objective**

  Visualize the maze grid.

- **Details**

  Implement a method to draw each cell of the maze on the Pygame window, differentiating walls from open spaces.Use different colors or graphics to clearly distinguish between these cell types.

2.4.9. Handling User Input:

- **Objective**

Allow users to interact with the program by setting the start and goal points.

- **Details**

Capture mouse clicks or keyboard events to set the start and goal points within the maze grid.Ensure the user interface is intuitive and responsive.

2.4.10. Visualizing the Pathfinding Process:

- **Objective**

Show the algorithm's progress in real-time.

- **Details**

Update the display to show explored nodes, current path considerations, and the final shortest path.Use distinct colors or markers for different stages of the algorithm’s execution.

2.5.11.Running the Main Loop:

- **Objective**

Integrate all components into the main application loop.

- **Details**

Continuously update the display, handle user input, and control the pathfinding process.Ensure smooth execution and real-time updates by maintaining a consistent frame rate.

2.4.12. Testing and Debugging:

- **Objective**

Ensure the correctness and robustness of the maze solver.

- **Details**

Test the maze solver with different maze configurations and sizes.Debug issues related to pathfinding accuracy, user input handling, and graphical updates.

2.4.13. Enhancing User Experience:

- **Objective**

Improve the overall usability and aesthetics of the application.

- **Details**

Add features like start, pause, and reset buttons.Include additional visual enhancements like animations, better graphics, and sound effects.

CHAPTER-3

3.PROGRAM

3.1.PSEUDOCODE:

```
Initialize required modules and constants
Define maze options and select one randomly
Convert selected maze to numpy array
Set tile size and calculate screen dimensions

Define heuristic function:
    Return Manhattan distance between two points

Define A* algorithm function:
    Initialize open list with start node
    Initialize parent, g_score, and f_score dictionaries
    While open list is not empty:
        Pop node with lowest f_score
        If node is end node, break
        Get neighbors of current node
        For each neighbor:
            Calculate tentative g_score
            If this path is better:
                Update parent, g_score, and f_score
                Push neighbor to open list
    Reconstruct path from end to start using parent dictionary

Define helper functions:
    Get valid neighbors for a given position in the maze
    Find and return start and end positions in the maze
```

Initialize Pygame and set up display

Define function to draw maze on screen
Define function to draw path and display messages
Define function to get direction of movement

Define main function:
    Initialize step and running flag
    While running:
        Handle Pygame events
        Draw maze and path
        Update display and increment step
        Control loop speed

Run main function if script is executed directly

3.2 CODING:

```
import numpy as np
import random as rand
from tkinter import messagebox
import pygame
import sys
from heapq import heappop, heappush
a = [
    ['S', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
    ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X',  ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', 'X',  ' ', 'X', ' ', 'X', 'X', 'X',  ' ',  ' ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', ' ', 'X',' ', ' ', ' ', 'X', ' '],
    ['X', ' ', 'X', ' ', ' ', 'X', '  ', '  ', '  ', '  ', '  ', ' ', ' ', ' ', 'X', ' ', 'X', 'X', 'X', 'X'],
    ['X', ' ', 'X',' ', 'X', 'X',' ','X', 'X', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', ' ', 'X'],
    ['X', ' ', 'X', '  ', 'X', '  ', '  ', '  ', '  ', '  ', '  ', '  ', 'X', ' ', 'X', ' ', 'X', ' ', ' ', 'X'],
    ['X',' ', 'X', ' ', 'X', ' ', ' ', 'X', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', 'X','X',' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', '  ', '  ', 'X', '  ', 'X', '  ', 'X', '  ', '  ', '  ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', ' ', 'X', 'X', 'X', ' ', 'X',' ', 'X',' ', 'X', 'X', 'X', 'X',' ', 'X'],
    ['X', '  ', 'X', '  ', '  ', '  ', 'X', '  ', '  ', '  ', 'X', '  ', 'X', ' ', ' ', ' ', ' ', 'X', ' ', 'X'],
    ['X','  ', 'X',' ', 'X', 'X', 'X',' ', 'X', 'X', 'X',' ', 'X',' ', 'X', 'X', ' ', 'X', 'X', 'X'],
    ['X', '  ', 'X', '  ', '  ', '  ', '  ', '  ', 'X', '  ', '  ', '  ', 'X', '  ', '  ', '  ', ' ', ' ', ' ', 'X'],
    ['X',' ', 'X',' ', 'X', 'X', 'X', 'X', 'X',' ','X', 'X','X', 'X','X','X', 'X', 'X',' ', 'X'],
    ['X', ' ', 'X', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', '  ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
     ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
     ['X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'E']
]
```

```
b = [
    ['S', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
    ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', 'X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', ' ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', 'X', 'X'],
    ['X', ' ', 'X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', 'X', 'X', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', 'X', 'X', 'X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', 'X', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'E']]

big_maze1 = [
    ['S', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
    ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', 'X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', ' ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', 'X', 'X'],
    ['X', ' ', 'X', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', 'X', 'X', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X', ' ', 'X', ' ', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', 'X', 'X', 'X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', 'X', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'E']]
```

```
    ['X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', 'X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', ' ', 'X', ' ', ' ', ' ', ' ', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', ' ', 'X', 'X', 'X', ' ', 'X', ' ', 'X', 'X', ' ', 'X', 'X', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', 'X', ' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', ' ', 'X'],
    ['X', ' ', 'X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', 'X'],
    ['X', ' ', 'X', ' ', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
    ['X', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
    ['X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'E']
]

WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
RED = (255, 0, 0)
GREEN = (0, 255, 0)
BLUE = (0, 0, 255)
ORANGE = (255, 165, 0)

li = [a, b,big_maze1]
ran = rand.choice(li)
maze = np.array(ran)

TILE_SIZE = 30
MAZE_WIDTH = len(ran[0])
MAZE_HEIGHT = len(ran)
SCREEN_WIDTH = MAZE_WIDTH * TILE_SIZE
SCREEN_HEIGHT = MAZE_HEIGHT * TILE_SIZE
```

```
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

def a_star(maze, start, end):
    open_list = []
    heappush(open_list, (0, start))
    parent = {start: None}
    g_score = {start: 0}
    f_score = {start: heuristic(start, end)}

    while open_list:
        _, current = heappop(open_list)

        if current == end:
            break

        neighbors = get_neighbors(maze, current)
        for neighbor in neighbors:
            tentative_g_score = g_score[current] + 1
            if neighbor not in g_score or tentative_g_score < g_score[neighbor]:
                parent[neighbor] = current
                g_score[neighbor] =tentative_g_score
                f_score[neighbor] = tentative_g_score + heuristic(neighbor, end)
                heappush(open_list, (f_score[neighbor], neighbor))

    path = []
    step = end
    while step:
        path.append(step)
        step = parent.get(step)
    path.reverse()
    return path
```

```
def get_neighbors(maze, pos):
    neighbors = []
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    for direction in directions:
        new_pos = (pos[0] + direction[0], pos[1] + direction[1])
        if 0 <= new_pos[0] < len(maze) and 0 <= new_pos[1] < len(maze[0]) and
maze[new_pos[0]][new_pos[1]] != 'X':
            neighbors.append(new_pos)
    return neighbors

def find_start_end(maze):
    start = end = None
    for i in range(len(maze)):
        for j in range(len(maze[0])):
            if maze[i][j] == 'S':
                start = (i, j)
            elif maze[i][j] == 'E':
                end = (i, j)
    return start, end

start, end = find_start_end(maze)
path = a_star(maze, start, end)



pygame.init()
pygame.display.set_caption("A* Maze Pathfinding")
SCREEN = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
CLOCK = pygame.time.Clock()
FONT = pygame.font.Font(None, 36)

def draw_maze():
```

```
    SCREEN.fill(WHITE)
    for i in range(len(maze)):
        for j in range(len(maze[0])):
            if maze[i][j] == 'X':
                pygame.draw.rect(SCREEN, BLACK, (j * TILE_SIZE, i * TILE_SIZE, TILE_SIZE,
TILE_SIZE))
            elif maze[i][j] == 'S':
                text = FONT.render('S', True, GREEN)
                SCREEN.blit(text, (j * TILE_SIZE + 5, i * TILE_SIZE + 5))
            elif maze[i][j] == 'E':
                text = FONT.render('E', True, RED)
                SCREEN.blit(text, (j * TILE_SIZE + 5, i * TILE_SIZE + 5))
way=[]
def draw_path(path, step):
    cost = 0
    for pos in path[:step]:
        if maze[pos[0]][pos[1]] not in ('S', 'E'):
            text = FONT.render('*', True, BLUE)
            SCREEN.blit(text, (pos[1] * TILE_SIZE + 5, pos[0] * TILE_SIZE + 5))
            cost += 1
        if maze[pos[0]][pos[1]] == 'E':
            pygame.display.flip()
            pygame.time.wait(1000)
            message = f"The goal state is reached\nThe total path cost is {cost + 1}"
            messagebox.showinfo(message=message)
            messagebox.showwarning(message="->".join(way))
            print(message)
            pygame.quit()
            sys.exit()
    if step < len(path):
        pos = path[step]
        text = FONT.render('X', True, ORANGE)
```

```
        SCREEN.blit(text, (pos[1] * TILE_SIZE + 5, pos[0] * TILE_SIZE + 5))
        if step > 0:
            prev_pos = path[step - 1]
            direction = get_direction(prev_pos, pos)
            if direction:
                message = f"Moving {direction}"
                pygame.display.set_caption(message)
                way.append(message)
                print(message)  # Print direction to console
        return cost

def get_direction(prev_pos, current_pos):
    if prev_pos[0] < current_pos[0]:
        return "down"
    elif prev_pos[0] > current_pos[0]:
        return "up"
    elif prev_pos[1] < current_pos[1]:
        return "right"
    elif prev_pos[1] > current_pos[1]:
        return "left"
    else:
        return None
def main():
    step = 0
    running = True
    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
        SCREEN.fill(WHITE)
        draw_maze()
        cost = draw_path(path, step)
```

```
        pygame.display.flip()
        step += 1
        CLOCK.tick(3)

    pygame.quit()

if __name__ == "__main__":
    main()
```

CHAPTER-4

4.RESULT

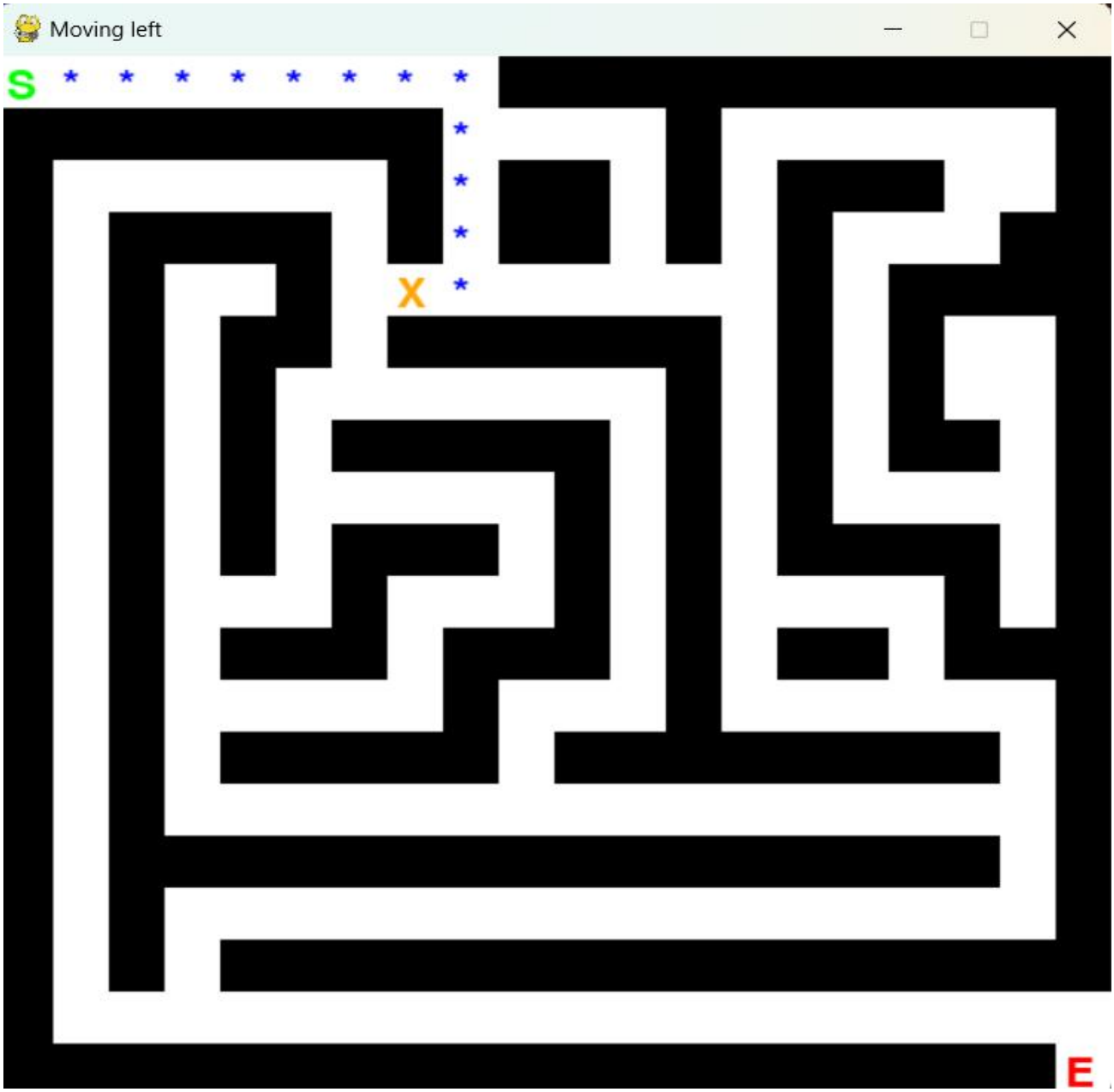

Figure No: 4.1.Before Reached (E)

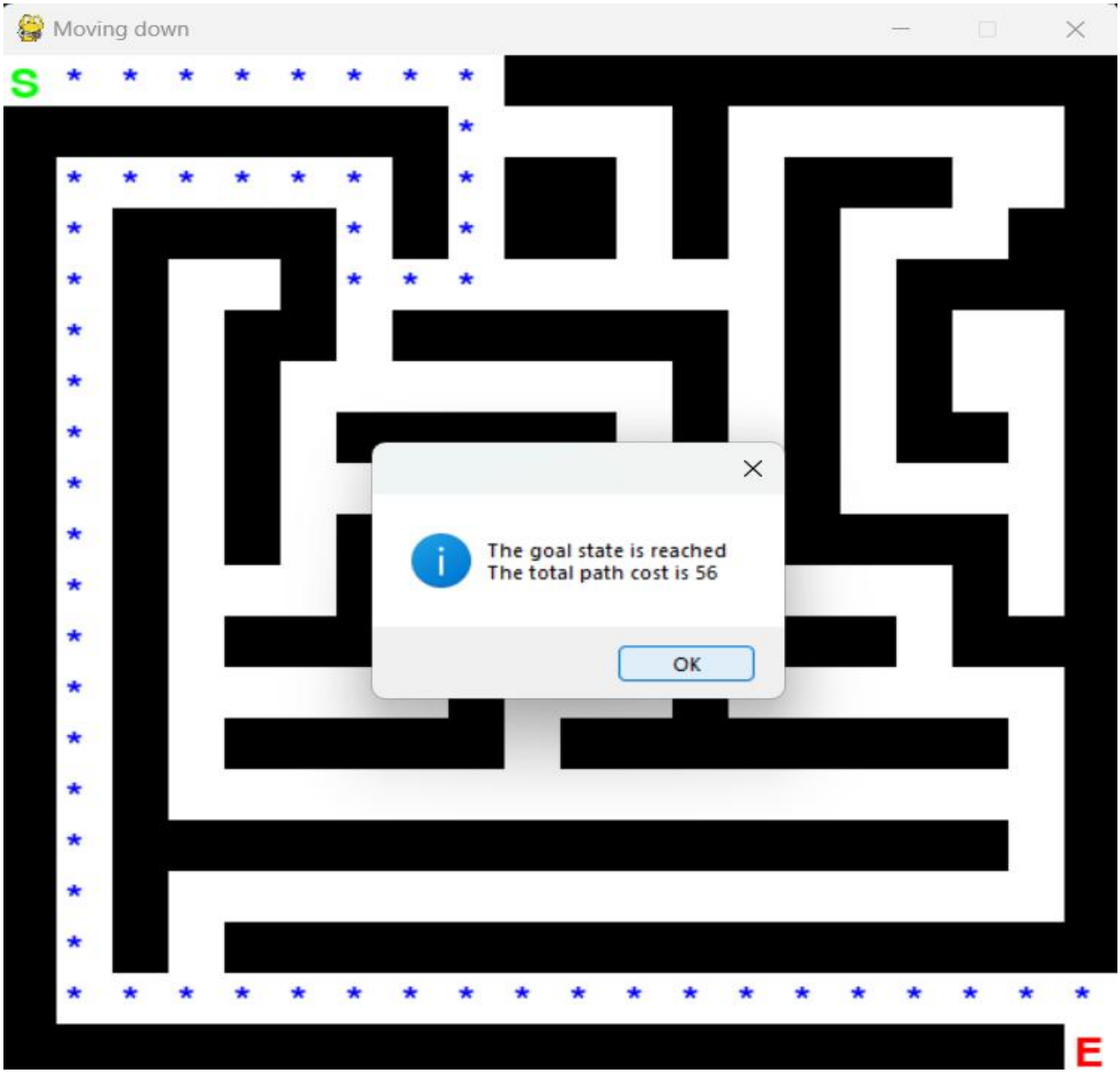

Figure No: 4.2.After Reached(E)

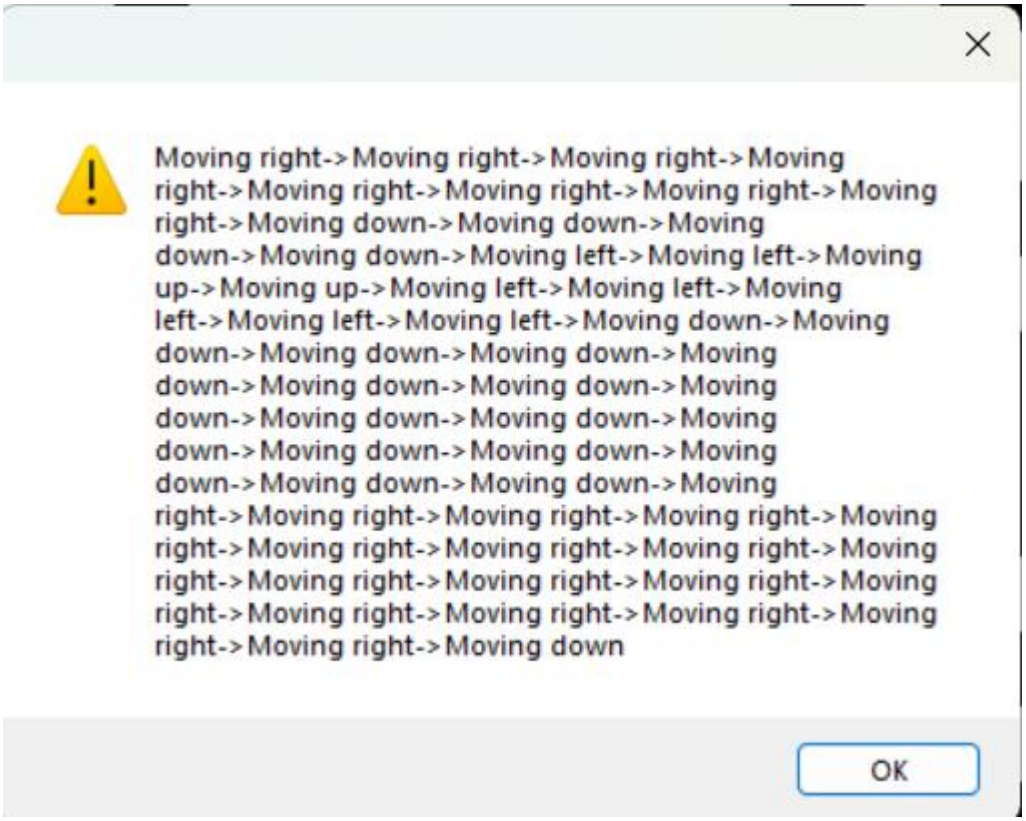

Figure No: 4.3.movements

CHAPTER-5

5.CONCLUSION

The A* search algorithm combines the strengths of Dijkstra's algorithm and greedy best-first search. It uses a heuristic to estimate the cost to reach the goal, enabling it to find the shortest path more efficiently.The heuristic function, specifically the Manhattan distance in this implementation, guides the search towards the goal while considering the actual cost from the start node. A* guarantees finding the shortest path if the heuristic is admissible (never overestimates the actual cost). In this maze runner problem, the Manhattan distance heuristic ensures optimality given that it accurately represents the minimum possible cost to reach the goal in a grid.

CHAPTER-6

6.REFERENCE

1. **https://www.wikipedia.org**
2. **https://theory.stanford.edu**
3. **https://www.geeksforgeeks.org**
4. **https://www.redblobgames.com**
5. **https://ocw.mit.edu**
6. **https://www.coursera.org**
7. **https://www.khanacademy.org**
8. **https://www.pythonpool.com**
9. **https://www.programiz.com**
10. **https://www.tutorialspoint.com**