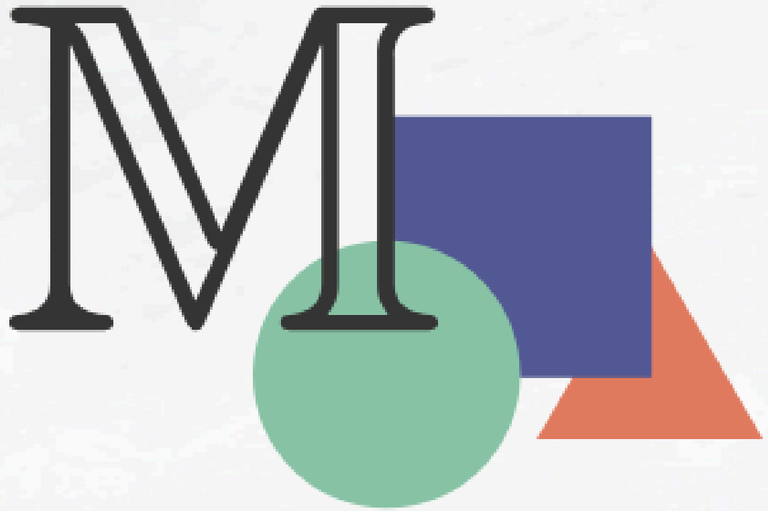
GROUP 5 - PROJECT

MANIMAZE

BACKTRACKING



Members

Trần Đàm Quốc Khánh

Nguyễn Duy Hưng

Trịnh Phương Anh

Nguyễn Trọng Hùng

Trần Ngọc Sơn

CONTENTS

1.INTRODUCTION

- Practical applications
- Background information
- Purpose of the project

MANIMAZE
BACKTRACKING

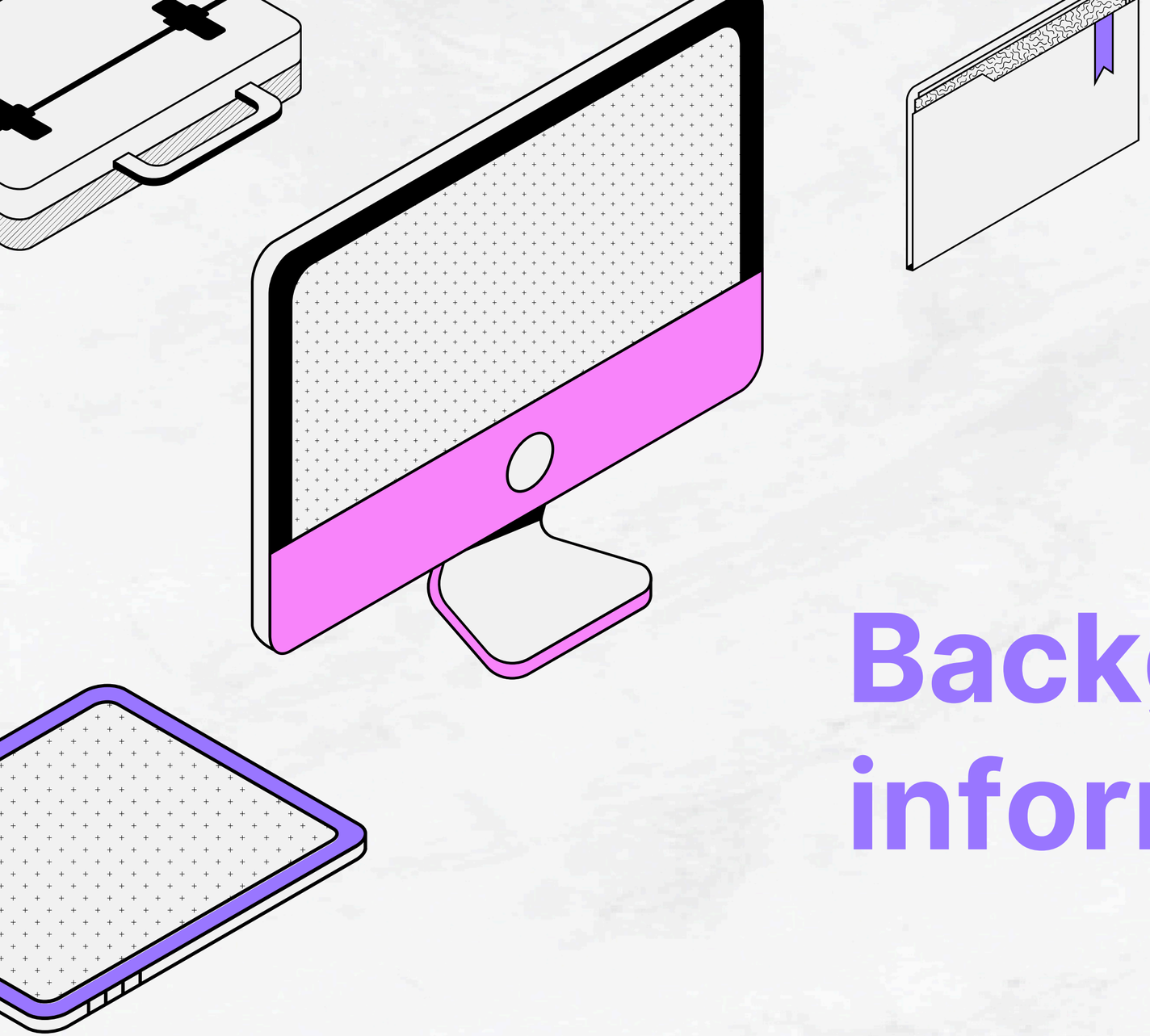
Practical applications

cleaning robot

- Detect obstacles, adjust paths to prevent collisions, ensuring effective cleaning in cluttered environments.
- Explore and map unknown areas efficiently.

WAIT BUT WHY

we will explain it in our project



Background information

meaning of the project's name

MANIMAZE BACKTRACKING

Manim

a versatile tool that can transform abstract mathematical concepts into visually engaging animations.

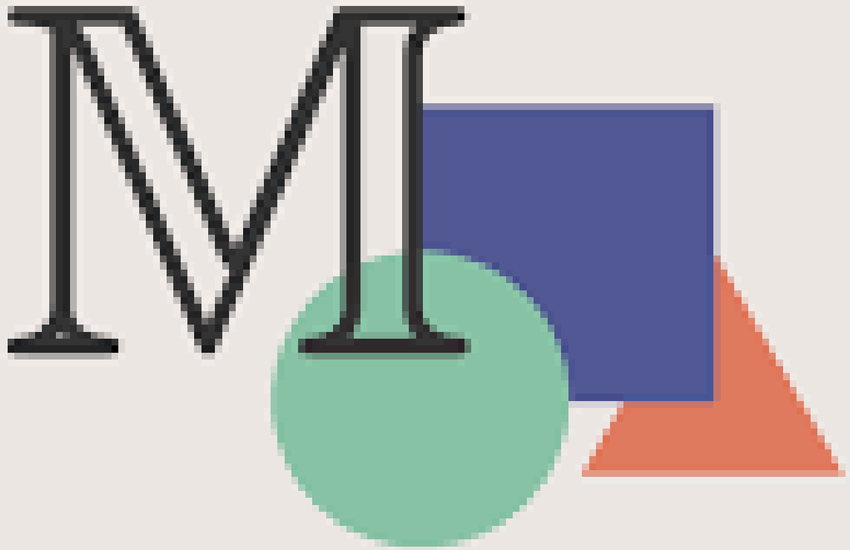
Maze

A complex structure with multiple paths and dead ends, serving as a backdrop for our exploration, where the objective is to navigate from the entrance to the exit.

Backtracking

Backtracking is a recursive problem-solving technique that incrementally builds a solution, removing failed solutions that do not meet the problem's constraints. It is used to navigate through mazes, making decisions and retracing steps.

In this project, Manim takes **center stage,** providing a dynamic and interactive visualization of the backtracking algorithm's journey through a maze.



purpose of the project

Manimaze Backtracking project showcases how **mathematics, programming, and visual storytelling** collaborate to make the abstract world of algorithms **more understandable** for users, both new and experienced.

Discover how Manim can transform **mathematical concepts and algorithms** into engaging **educational tools**.

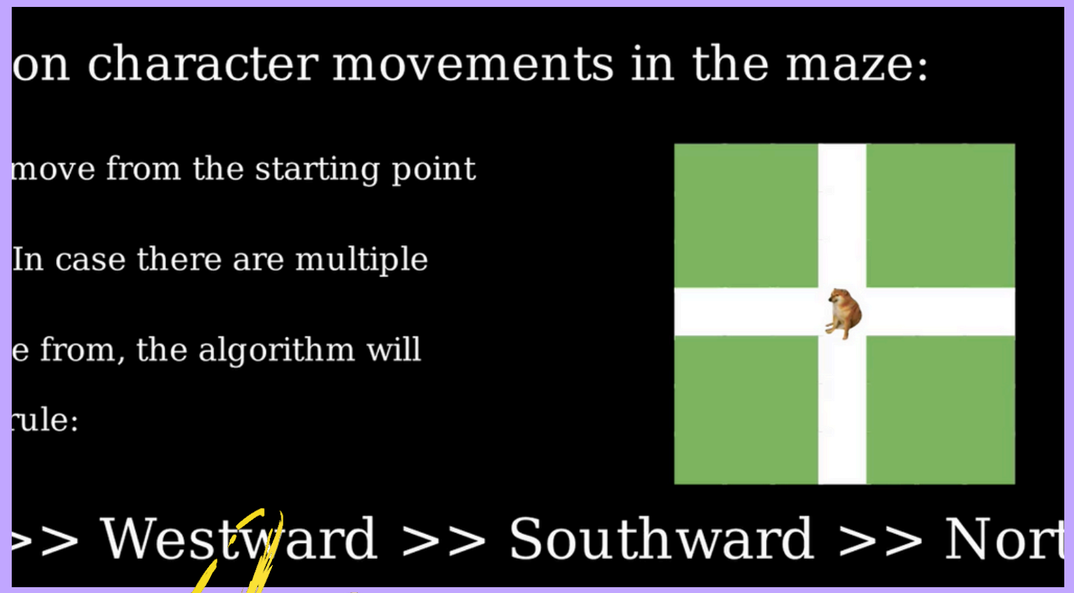
2. Project Description

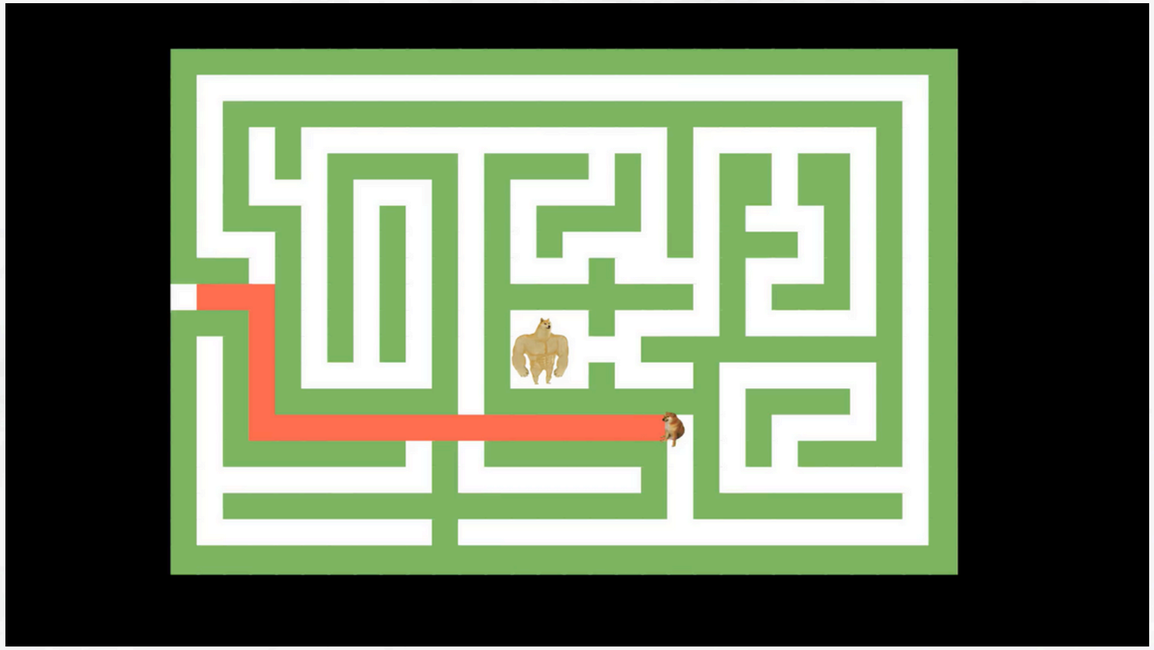
2.1. Design and development steps for the animations
2.2. Tools and techniques used
2.3. Algorithms and logic behind the animations
2.4. Challenges and difficulties faced

2.1.Design and development steps for the animations

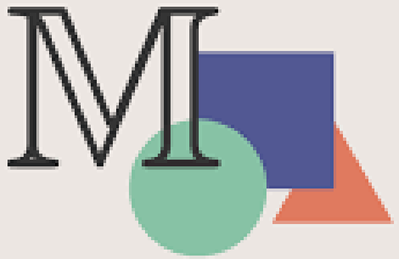
The project is made up of **different scenes**
-> show different aspects of maze traversal and algorithmic decision-making.

The animation consists of **text animations** and **three scenes**, including *one main scene* and *two sub-scenes*.





All of these scenes incorporate **a maze**, **a character**, and animations depicting the **character's movement within the maze**.



2.2. Tools and techniques used

2.3. Algorithms and logic behind the animations

a) Create maze using matrix

b) Deep first search (DFS)

a) Create maze using matrix

The maze is created by numerous small squares

-> Create a matrix with 0 refers to wall, 1 refers to path

```
[0, 0, 0, 0, 0, 0, 0 ]
[0, 1, 1, 1, 1, 1, 1 ]
[0, 1, 0, 0, 0, 0, 0 ]
[0, 1, 0, 1, 1, 1, 0 ]
[0, 1, 0, 1, 0, 0, 0 ]
[1, 1, 0, 1, 1, 1, 0 ]
[0, 1, 0, 0, 0, 1, 0 ]
[0, 1, 1, 1, 1, 1, 0 ]
[0, 0, 0, 0, 0, 0, 0 ]
```

Nested Loop for Creating Squares

- The code uses a **nested loop** to iterate through each element in the maze_matrix.
- The outer loop (for i, row in enumerate(maze_matrix)) iterates through the rows
- The inner loop (for j, val in enumerate(row)) iterates through the columns.

```
for i, row in enumerate(maze_matrix):
    maze_row = []
    for j, val in enumerate(row):
        if val == 0:
            square = Square(side_length=square_size, color=GREEN, fill_opacity=1).shift(
                ((j - len(row) / 2) * square_size + len(row) * square_size) * RIGHT + (len(maze_matrix) / 2 - i) * square_size * UP
            )
        elif val == -1 :
            square = Square(side_length=square_size, color=GREEN, fill_opacity=1).shift(
                ((j - len(row) / 2) * square_size + len(row) * square_size) * RIGHT + (len(maze_matrix) / 2 - i) * square_size * UP
            )
        else:
            square = Square(side_length=square_size, color=WHITE, fill_opacity=1).shift(
                ((j - len(row) / 2) * square_size + len(row) * square_size) * RIGHT + (len(maze_matrix) / 2 - i) * square_size * UP
            )
        maze_row.append(square)
        self.add(square)

    maze.append(maze_row)
```

Instructions on character movements in the maze:

The character will move from the starting point

to the destination. In case there are multiple

directions to choose from, the algorithm will

follow the priority rule:

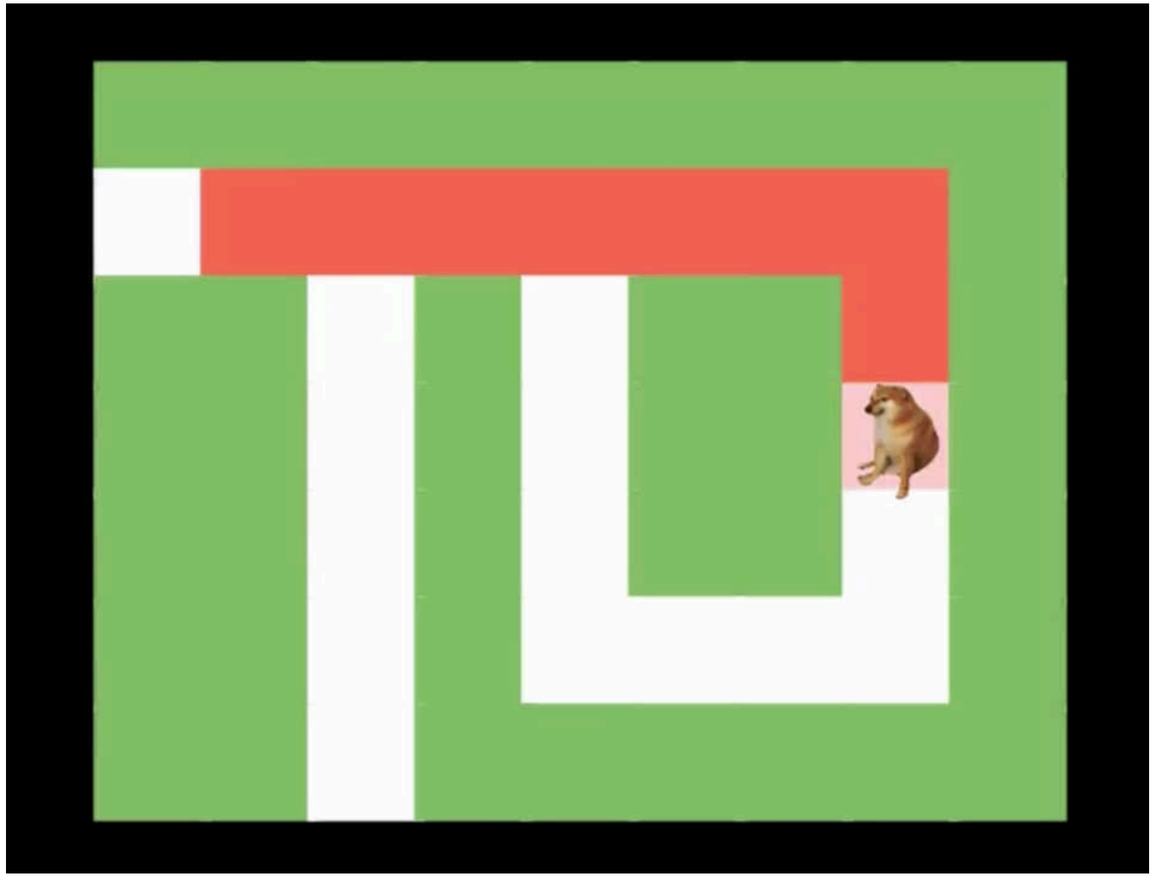
Eastward >> Westward >> Southward >> Northward.

b) Deep first search (DFS)

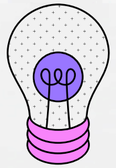
- DFS is a graph traversal algorithm that can be adapted for maze-solving
- Step:
  - Start DFS from the entrance of the maze
  - Move as deeply as possible along each path (East - West - South - North)
  - If a dead end is reached, backtrack to the previous cell and explore other directions.

b) Deep first search (DFS)

- Also backtrack the character if it goes to the cell it has passed through
- Continue the process, return True if character position is the end point and return False if there no more feasible path to go

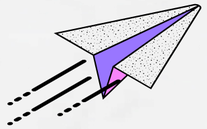


III. Conclusion



Tools and Techniques

1) Primary tool Manim
2) Python knowledge pivotal
3) Incorporates mathematical logic



Algorithms and Logic Behind Animations

1) Backtracking
2)Matrix
3)Depth-First Search

Challenges Faced

1) Initial installation
2) Maze matrix visualization complexity
3) Understanding and implementing the (DFS) algorithm

Manimaze Backtrac_