**NATIONAL ECONOMICS UNIVERSITY**

**FACULTY OF ECONOMIC MATHEMATICS**



**PROJECT - MANIM**

**TOPIC: MANIMAZE BACKTRACKING**

Class : DSEB 64A

Instructor : Dr. Nguyễn Tuấn Long

Group members : Trần Ngọc Sơn – 11225650 – 20%

Trần Đàm Quốc Khánh – 11223079 – 20%

Nguyễn Duy Hưng – 11222611 – 25%

Nguyễn Trọng Hùng – 11222582 – 15%

Trịnh Phương Anh – 11220655 – 20%

**HANOI, 2023**

**TABLE OF CONTENTS**

**I.** **Introduction : Manimaze Backtracking** 1

1. **Brief background of the project and purpose of the report** 1

2. **Introduction to Manim** 1

**II.** **Main Content** 2

**1.** **Description of project implementation process using Manim** 2

**1.1. Design and development steps for the animations** 2

**1.2. Tools and techniques used:** 3

**1.3. Algorithms and logic behind the animations** 3

**1.4. Challenges and difficulties faced:** 7

**2. Evaluation of using Manim** 7

**2.1. Advantages** 7

**2.2. Disadvantages** 8

**2.3. Comparison with other tools** 8

**III.** **Conclusion** 9

2. **Introduction : Manimaze Backtracking**
3. **Brief background of the project and purpose of the report**

"Manimaze Backtracking" is a captivating project that seamlessly blends the power of the Manim animation engine with the intricacies of maze-solving through the backtracking algorithm. In this project, we utilize Manim's dynamic capabilities to visually represent the step-by-step exploration of a maze using the backtracking technique.

The name "Manimaze Backtracking" encapsulates the essence of the project:

* Manim: Refers to the Mathematical Animation Engine, a versatile tool known for its ability to transform abstract mathematical concepts into visually engaging animations.
* Maze: Represents the challenging labyrinthine structures that serve as the backdrop for our exploration.
* Backtracking: an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point in time. In our specific context, backtracking is applied to systematically navigate through the maze, 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. The end result is an engaging and instructive journey that brings the abstract world of algorithms to life, making it more understandable for both new and experienced users. This Manimaze Backtracking project is a good example of how mathematics, programming, and visual storytelling work together.

1. **Introduction to Manim**

In the realm of mathematical and scientific animation, Manim, short for Mathematical Animation Engine, stands as a powerful and versatile tool that transcends traditional methods of visualizing complex concepts. Manim is an open-source Python library renowned for its ability to transform abstract mathematical ideas into captivating visual narratives.

During our exploration of "Manimaze Backtracking," it becomes essential to grasp the significance of Manim in the context of our project. Manim not only provides a dynamic canvas for translating code into animated sequences but also facilitates an immersive learning experience by making intricate algorithms, such as backtracking, accessible and engaging. The extensibility and customization options of Manim allow us to tailor the visualization to the specific nuances of our maze-solving algorithm, enabling a seamless fusion of mathematical rigor and visual storytelling.

In this section, we delve into the features that make Manim an ideal choice for our project, emphasizing how it serves as a conduit for bridging the gap between abstract algorithms and tangible, animated representations. Through the lens of "Manimaze Backtracking," we unravel the transformative potential of Manim as an educational instrument, showcasing its ability to breathe life into complex mathematical concepts and algorithms.

1. **Main Content**
2. **Description of project implementation process using Manim**

**1.1. Design and development steps for the animations**

The provided code shows how to use the Manim library to make animated visualizations of maze-solving algorithms using Python. The project is made up of different scenes that show different aspects of maze traversal and algorithmic decision-making.

The foundation of the project lies in the representation of mazes through matrices, where 0s delineate walls, and 1s signify navigable paths. The scenes are delineated into distinct entities, with the Huong scene exemplifying the character's movement within the maze via a depth-first search algorithm. On the other hand, the Backtrack\_tutorial scene elucidates the intricacies of backtracking when the character encounters obstacles or reaches a dead-end.

To bring these scenes to life, the code introduces functions such as maze\_tutorial, specially crafted to initialize and animate mazes based on predefined matrices. These functions seamlessly incorporate the character's movement, whether it be the strategic exploration of pathways using depth-first search or the nuanced backtracking process.

The orchestration of scenes is streamlined through the CombinedScene class, weaving together diverse elements into a coherent and visually compelling narrative. This class adeptly iterates through each scene, unveiling their construction, and then gracefully fading out elements to facilitate smooth transitions between segments.

In essence, this project unfolds as a synergy between code, mathematics, and visual storytelling. The amalgamation of Manim, Python, and geometric principles converges to deliver an educational and immersive exploration of maze-solving techniques. Through animated vignettes, viewers are invited to traverse the intricate landscapes of algorithms, unlocking a deeper understanding of these mathematical constructs in an engaging and accessible manner.

**1.2. Tools and techniques used:**

In the implementation of this project, the primary tool utilized is Manim, a powerful Python library designed for creating mathematical animations. Manim serves as the core engine for generating visually engaging content, and a solid understanding of the Python programming language is pivotal, given that Manim is built upon Python.

The creative process involves leveraging geometric principles and shapes. Various geometric elements, such as squares and circles, are employed to represent maze structures, obstacles, and the character's movement. Furthermore, geometric transformations play a crucial role in animating these elements, allowing for dynamic and visually appealing representations.

The project showcases how Python, when coupled with Manim, becomes a potent tool for expressing complex mathematical concepts through animated visuals. The code incorporates mathematical logic to simulate maze traversal and backtracking algorithms, contributing to an interactive and educational presentation.

**1.3. Algorithms and logic behind the animations**

1. **Maze Matrices:**

Full Maze: The primary maze matrix consists of 20 rows and 30 columns, with '0' representing open paths, '1' indicating the goal, and the absence of any value implying obstacles. This matrix forms a complex maze structure with multiple paths and dead ends.

Maze 1: A condensed version of the full maze, `maze\_matrix1`, is presented with dimensions 9x7. It captures the essence of the larger maze, maintaining the same conventions for '0', '1', and obstacles. This reduced matrix serves as a concise representation for specific scenarios or focused analysis.

Maze 2: Another reduced maze, 'maze\_matrix2', measures 7x9. It exhibits a different layout from `maze\_matrix1`, providing diversity in maze structures. Similar to the full maze, it follows the '0', '1', and obstacle conventions, offering alternative challenges for analysis or educational purposes.

Mazes 4, 5, and 6, represented by maze\_matrix4, maze\_matrix5, and anmaze\_matrix6, respectively, feature distinct 7x7 configurations. Maze 4 challenges navigation algorithms with a unique layout, while Maze 5 invites the exploration of varied strategies. Maze 6 intensifies complexity with increased obstacle density, testing the adaptability of maze-solving algorithms. These matrices collectively contribute to a diverse set of scenarios for algorithmic analysis and educational animations, fostering dynamic and engaging learning environments.

These maze matrices serve as foundational elements for creating diverse and visually appealing maze animations. The variations in size and structure enable flexibility in designing scenarios for educational or entertainment purposes. Further development may involve incorporating animations and algorithms for maze-solving within these defined structures.

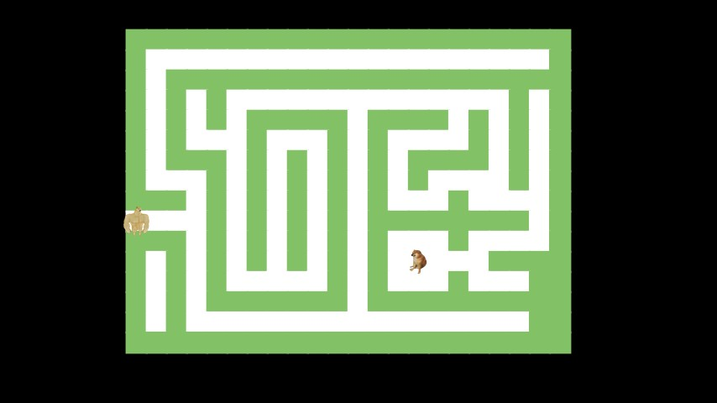
1. **Create maze:**

The initial part of the code establishes parameters for scaling, with a scale\_factor of 0.8 (0.65 for the final example) and a calculated square\_size set at 0.4. These parameters will influence the size of the squares in the maze.

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, and the inner loop (for j, val in enumerate(row)) iterates through the columns.

Square Creation: For each element (val) in a row, a Square object is created. If val is 0, indicating an open path, a green square is created with the specified side\_length and shifted to its position in the scene. The shift function utilizes Manim's geometric transformations to place the square accurately within the maze. The (j - len(row) / 2) \* square\_size + len(row) \* square\_size + 1 term determines the x-coordinate, and (len(maze\_matrix) / 2 - i) \* square\_size determines the y-coordinate. If val is not 0, indicating an obstacle or goal, a white square is created using similar logic.

Adding Squares to the Scene: Each square is added to the Manim scene using self.add(square), contributing to the overall visualization of the maze. Each row of squares is appended to the maze list, creating a 2D representation of the maze.



2D Maze created by this code

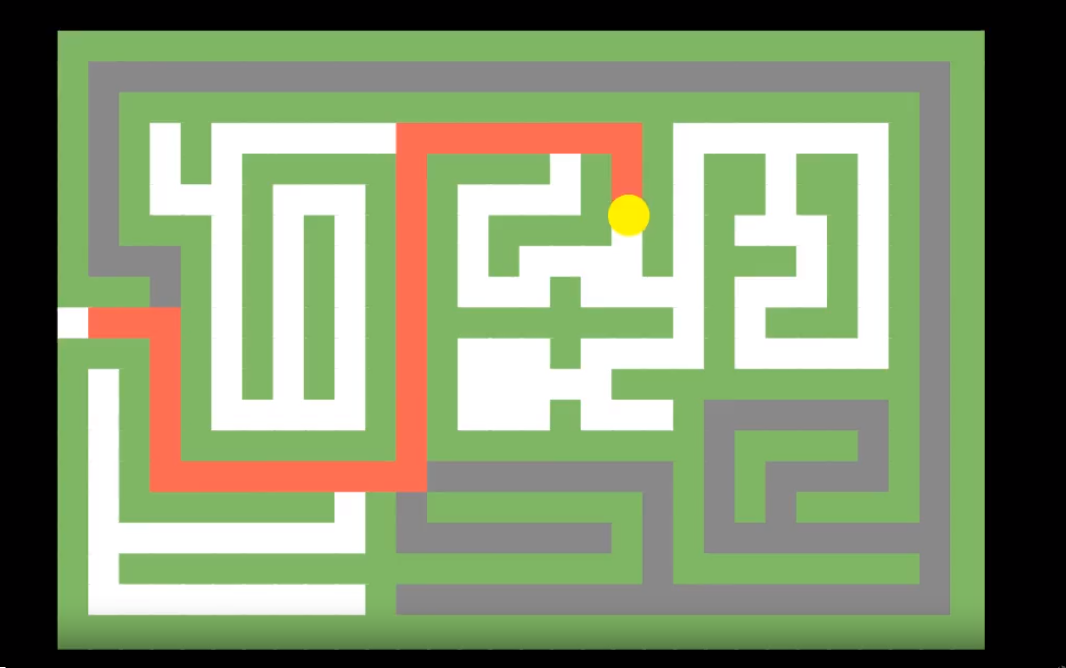
1. **Deep first search:**

DFS Initialization (dfs(start[0], start[1])): The DFS algorithm is initiated with the starting coordinates.

DFS Algorithm (dfs function):

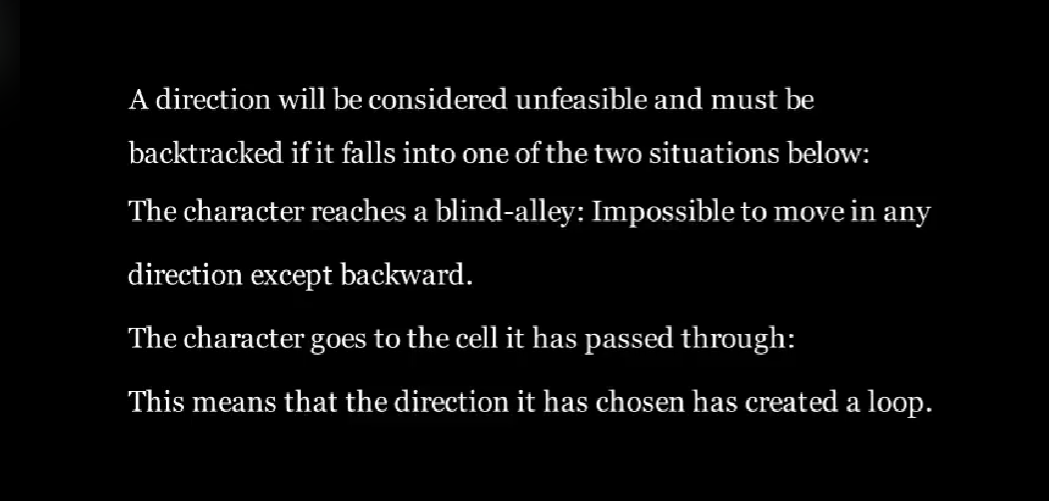
* Path List: The path list is used to keep track of the visited cells during the DFS traversal. It starts with the initial position (x, y).
* Base Case: If the current position (x, y) is the exit (end), the DFS algorithm returns True.
* Marking Visited: The current position is marked as visited in the visited matrix.
* Iterating Through Neighbors: The algorithm iterates through the possible directions (up, down, left, right) using the for loop. For each valid and unvisited neighbor, the algorithm updates the path, marks the cell as visited, and animates the movement of moving\_object to the new position.
* Color Changes: The corresponding square in the maze is colored red (maze[new\_x][new\_y].animate.set\_color(RED)) to visually represent the visited path.
* Recursive Call: The DFS algorithm is called recursively for the neighbor.
* Backtracking: If DFS from the neighbor does not lead to the exit, the color of the square is reset to gray (maze[new\_x][new\_y].animate.set\_color(GRAY)) to visually represent backtracking. The last position is popped from the path list.
* Return False: If no valid path is found from the current position, the function returns False.

Visualization Updates During DFS: Throughout the DFS execution, the movement of moving\_object and color changes in the maze squares (red for visited, gray for backtracked) are visually represented using self.play statements.

 The animation in the process of solving the maze

1. **Text:**

A series of text objects are created to provide explanations for the animations. Each text object is aligned to text1 to maintain a consistent left alignment. This ensures a neat and organized presentation. The self.play(Write(text)) statement is used to animate the appearance of each text object on the screen sequentially. This provides a step-by-step explanation of the algorithm.



Simple text to provide explanations

**1.4. Challenges and difficulties faced:**

The process of downloading and installing Manim posed challenges related to dependency management and compatibility, as it required careful consideration of system configurations and Python versions.

The maze matrix construction presented difficulties in visualizing the matrix effectively using Manim, with complexities arising from designing a clear representation and initializing the matrix with appropriate values.

Understanding and implementing the Depth-First Search (DFS) algorithm introduced challenges in grasping the recursive nature of the algorithm and integrating it seamlessly with visual animations using Manim.

**2. Evaluation of using Manim**

**2.1. Advantages**

Manim library provides us a high level of customization, allowing us to tailor the animations to our liking. From colors to styles, this flexibility ensures that the final product not only conveys information effectively but also does so in a visually appealing manner, contributing to the overall project aesthetics.

Manim's standout feature lies in its ability to create visually captivating animations. By tapping into Manim's prowess, we can dynamically visualize the maze generation and recursive backtracking process, adding an extra layer of engagement and clarity to our project. The animated representation serves as a powerful educational tool, aiding in a more intuitive understanding of the intricate algorithm.

A thriving community surrounds Manim, offering a valuable support system. The abundance of resources, including documentation, tutorials, and community forums, provides a robust foundation for project development. Leveraging this collective knowledge can help overcome challenges, share insights, and optimize our use of the Manim library.

**2.2. Disadvantages**

Manim poses a challenge with its steep learning curve, especially for those unfamiliar with the library. Investment in understanding Manim's nuances may translate to a time-consuming hurdle in our project timeline. In fact, our project group had to spend half of our project time just to install Manim and research about this library

Debugging animations introduces an additional layer of complexity compared to traditional code. Tackling issues in both Manim-specific code and the underlying mathematical algorithms demands a nuanced debugging approach, adding intricacy to our development process.

The creation of intricate animations with Manim demands substantial resources. Depending on the scale of our maze and animation, we may encounter performance issues and prolonged rendering times, a consideration particularly crucial for projects with resource constraints.

**2.3. Comparison with other tools**

Because in this semester the project team has also been exposed to the Matplotlib library which has a pretty similar functionality to Manim so we will compare these two libraries:

When it comes to visual sophistication and flexibility, Manim, conceived for mathematical visualizations, emerges as a powerhouse in crafting high-quality, visually sophisticated animations. Its forte lies in granting users meticulous control over each frame, facilitating the creation of intricate graphics. Moreover, Manim's support for three-dimensional graphics elevates it to an ideal choice for projects where an artistic touch is imperative. However, the advantages of Manim come with a trade-off; its steep learning curve and intricate syntax pose challenges, potentially hindering those seeking a swift and practical solution.

In contrast, Matplotlib, while not wielding the same level of artistic finesse as Manim, shines in its simplicity. Offering clear and concise visualizations, Matplotlib is accessible for educational purposes and quick prototyping. Yet, it falls short in replicating the intricate details achievable with the more artistically advanced Manim.

Sum up, Manim offers unparalleled artistic sophistication but requires an investment in time and effort to master. Matplotlib, with its simplicity and accessibility, caters to swift solutions but may compromise on the intricate details achievable with Manim. The choice between Manim and Matplotlib hinges on the project's specific requirements. However, given the complexity of the current problem chosen by the project team, we thought it would be easier to create the product with the Matplotlib library but Manim would be a better choice if we wanted to improve and develop this project in the future

1. **Conclusion**

With six weeks to complete the project, our team spent three weeks learning background knowledge about the Manim library and researching to find the appropriate topic, approach methods as well as development options. In the remaining time, it took us the following week to write and run the algorithm code and create the main scene, the next week was used to add text and create sub-scenes illustrating the solution to the problem. The remaining time is spent writing reports and preparing presentation slides. All steps in the project process involve the participation, contribution and agreement of every member in the group.

We chose to illustrate recursion backtracking because it is an attractive topic that can be a solution to complicate problems without worrying too much about algorithms. The direction of the maze was chosen since it has the appropriate level of difficulty, is easy to simulate when working in two dimension graphics, and is quite applicable in real life. This is also an approach that can have many potential developments in the future.

The products of this project are a video finding feasible path in a maze to explain how recursion backtracking works, a project report, and presentation slides. Moreover, throughout this project, our team members had the opportunity to get access and understand more about the Manim library, and gain more experience in working on a code project.

If we have more understanding, more time to pursue this project, or if we do this project again in the future, we would consider optimizing the running time of the algorithm in order to be able to find the shortest path for the maze. Researching how to make video by three-dimensional graphics while creating a themed maze and adding effects to the scene will be good choices to develop this project.