Maze Solving using Flood Fill Algorithm

Rayan Bouhal and Niko Paraskevopoulos

October 2023

Introduction to Micromouse

In the world of Robotics there is a competition called Micromouse. Micromouse uses Robotics, Computer Science, and Combinatorics to create an autonomous mouse that is no bigger than $16cm \times 16cm$ which can plan its own path to solve a $2.88m \times 2.88m$ maze. The mouse that finds the center of the maze in the shortest time wins. [7].

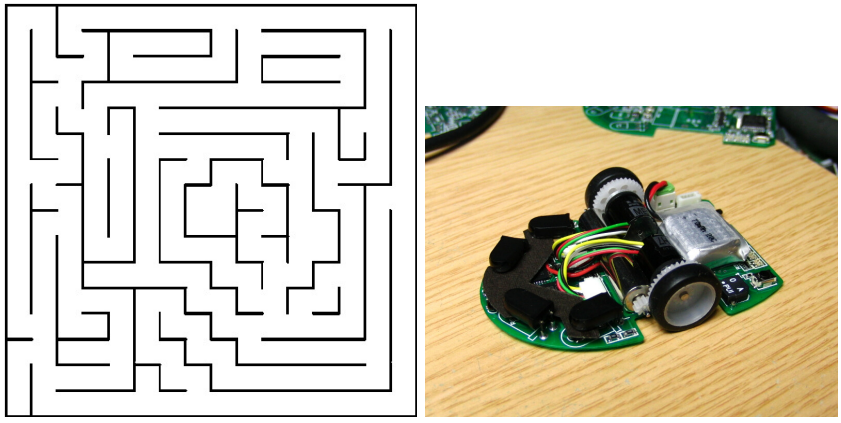


Figure 1: Example Micromouse maze and mouse [7][2]

Core Concepts

Robotics

The true feat of Micromouse competitions lies in the remarkable design achievement of constructing hardware that adheres to stringent size requirements while allowing the mouse to execute intricate maneuvers, including right, left, and diagonal turns. This integration of hardware is particularly challenging as it involves the meticulous miniaturization of complex components like sensors, processors, and motor controllers, which must be small and light, yet robust

enough for rapid and precise maze navigation. These miniature marvels employ infrared sensors which map the maze in a limited number of runs, and then deploy their path finding algorithms to race to the center of the maze. Typically, the fastest route is also the shortest path; however, depending on a mouse's hardware configuration, exceptions may arise. In one memorable competition, the winning mouse traversed a path that deviated from its competitors. This path, technically longer, featured fewer 90° turns, allowing the mouse to achieve higher speeds despite its extended route.

While the hardware component of the competition is undeniably pivotal and presents its own set of challenges, this research paper will exclusively focus on the software and mathematical concepts of Micromouse. It is worth noting that without the hardware component, the Micromouse challenge would not be complete.

Computer Science

Another core concept at work in Micromouse is the software design of an intelligent decision making algorithm. There are many algorithms that have been used to solve a Micromouse maze, the most popular are Flood Fill, Dijkstra's, A*, and wall-following [4][3]. Flood Fill, Dijkstra's, and A* are advanced pathfinding algorithms, each with a unique approach to navigating mazes: Flood Fill systematically fills the maze to determine the shortest path, Dijkstra's algorithm finds the shortest path between nodes, and A* efficiently combines features of both for optimal pathfinding. On the other hand, wall-following is a simpler method where the mouse follows the wall of the maze, often resulting in longer paths. Currently, the best and most popular algorithm used in competitions is Flood Fill, with Dijkstra's and A* being slightly behind, and wall-following being considered the poorest algorithmic choice. The development of these algorithms has taken 30 years to perfect for this challenge [4].

In this research project we will present a visualization of the Flood Fill Algorithm that solves a randomly generated Micromouse maze, to gain a deeper understanding of the computing and mathematical concepts introduced by Micromouse. [1].

Combinatorics

All of the algorithms needed to solve Micromouse are derived from combinatorial concepts with the main concepts being graph theory, shortest paths, and network algorithms. In the following section we will present an overview of the Flood Fill Algorithm by defining the necessary combinatorial contexts.

Combinatorial Definitions

Network

A network can be described as a graph where each edge is assigned a nonnegative integer [6]. This value, denoted by k, where $k \in Z$ and $k \geq 0$, essentially

represents the weight, length, or cost associated with traversing an edge within the network. An example of such a network is depicted in the figure below, where the k values on each vertex indicate these characteristics.

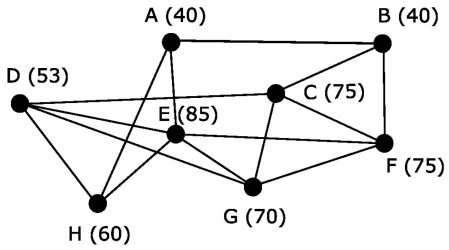


Figure 2: Illustration of a Network

The significance of these k values lies in their utility in modeling various real-world scenarios, where they can represent distances, costs, or other relevant metrics [6].

Shortest Path

Shortest paths are a fundamental aspect of network optimization [6]. In these problems, the goal is often to find the most efficient route between two points within a network. Given a start and a destination, the challenge is to determine the shortest path connecting these two vertices. Brute-force methods to solve this problem can be extremely time-consuming, especially in large graphs. Thus, shortest path algorithms are employed for a more efficient solution [6].

Among the various shortest path algorithms, our focus is on the Flood Fill algorithm. In the context of Flood Fill, the integer value k on each edge of the network represents the distance or proximity from the maze's exit. This approach is particularly effective in scenarios like maze-solving, where the objective is to navigate through a labyrinthine structure in the most efficient manner.

In essence, a maze can be viewed as an undirected and connected graph, where the objective is to identify a shortest path from a given vertex a to another vertex b. It's important to note that there could be multiple shortest paths, each influenced by various factors such as the layout of the maze and the specific algorithm used for pathfinding.

Application of Flood Fill Algorithm

Building on the foundational concepts of networks and shortest paths, let's delve deeper into the specific application of the Flood Fill algorithm. In a Flood Fill Network, the weights or k-values of the edges are determined based on their distance to the target destination. Unlike traditional representations of graphs with vertices and edges, the network in Flood Fill is more commonly visualized

as a grid or matrix. This representation aids in a clearer understanding and more intuitive visualization of the maze-solving process.

The following figure showcases an example graph of a grid. Here, the green circles represent the vertices, and the black lines between them signify the edges. This arbitrary graph can be transformed into a network using the Flood Fill algorithm, effectively illustrating how the algorithm navigates through a grid-like structure.

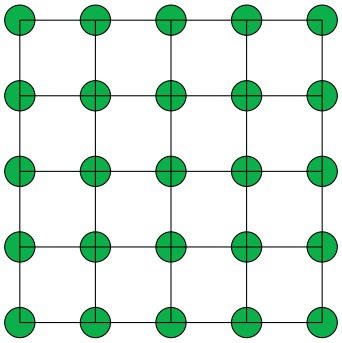
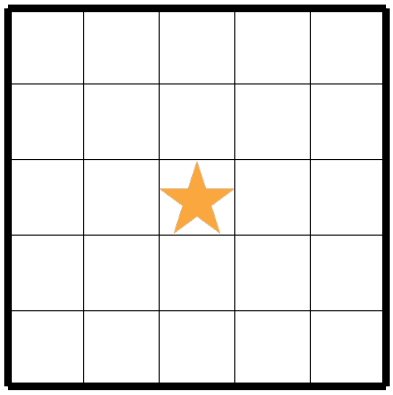


Figure 3: Grid representation of a graph for Flood Fill Network

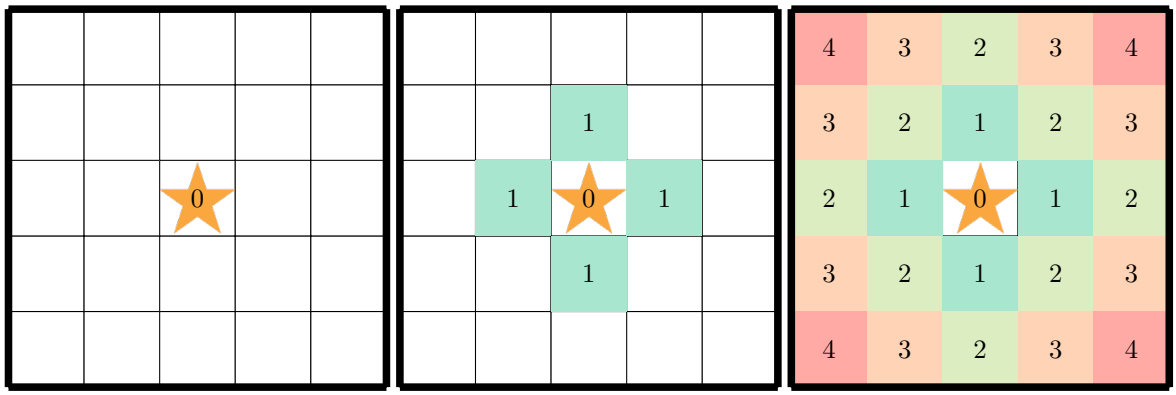
Note: For the remainder of this paper, our visualizations will utilize a grid format rather than a traditional graph. This approach is chosen for its clarity, simplicity in illustration, and adherence to established conventions.

Flood Fill Algorithm



We can visualise our maze as an $m \times m$ grid. Generally, for Micromouse $m = 16$. However, for simplicity we will reduce our m to 5. With the end of the maze being the center block, represented with a yellow star.

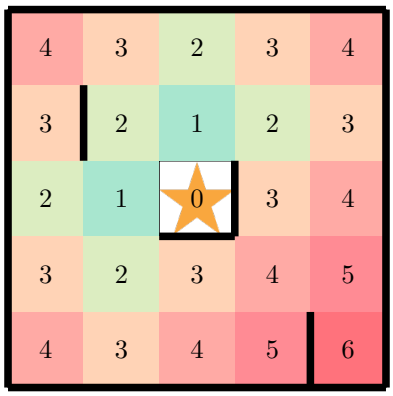
Proximity k-Values



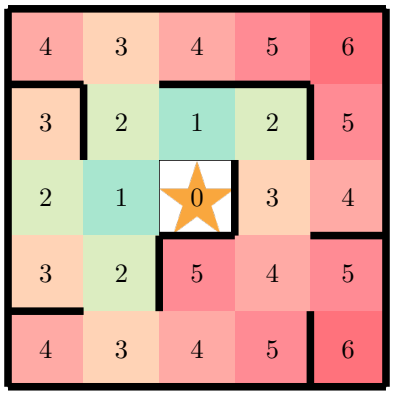
The first step of the flood fill algorithm is to assign numeric proximity values to all the squares in our grid. We previously defined this value as k.

1. **Identify Start and Destination:** Choose the start vertex and the destination vertex. Assign a weight of 0 to the destination.
    - *In the drawing above our destination is denoted by the yellow star and our start will be the bottom rightmost square.*
2. **Initialize Destination Edges:** Assign a weight of 1 to all edges directly connected to the destination vertex. These edges are highlighted in green.
    - *It is important to note that for our specific implementation, diagonal paths are not considered*
3. **Propagate Weights:** For each vertex reached by a weighted edge, assign a weight of 2 to all its unweighted edges, which are now highlighted in a yellow-green.
4. **Iterate and Assign Weights:** Continue the process of propagating the weights from vertices that have just been assigned a weight, incrementing the weight by 1 each time until all edges in the graph are weighted.

The initial mapping depicted above is how the mouse first views the maze. On the first iteration the mouse will act as if the maze has no walls. Then as the mouse runs through the maze and senses walls it recomputes the proximity values.



Here is a simple illustration to show, that by adding just a few walls the Flood Fill Algorithm completely recalculates the proximity values.

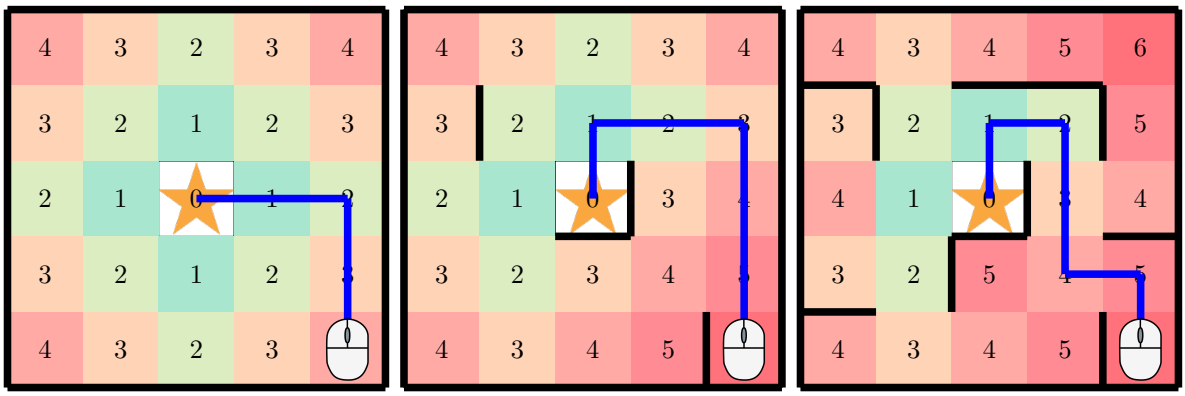


Here is a more complex maze that better imitates the Micromouse competition.

Shortest Path

Now that we have mapped the proximity k-values we are going to use those values to help us find the shortest path. Let's consider all three mazes we just viewed.

Let the starting position of the mouse be the bottom right corner facing the top of the maze. The Flood Fill Algorithm will find the shortest path by following descending proximity values. If it reaches a square where the values surrounding that square are equal, it will prioritize the straight path, since avoiding turns is generally faster for a vehicle.



Research Approach

We aim to develop a graphical simulation of a mouse navigating a maze using the Flood Fill Algorithm by integrating insights from various research papers. Komák and Pivarčiová employed CAD, a 3D modeling software, to visually represent the maze [1]. In contrast, we will utilize JavaScript—a language we are more proficient in—for both graphics and the algorithm's implementation.

Komák and Pivarčiová emphasized the need for the underlying code to detect maze walls and facilitate the mouse's movements—forward, left, and right [1]. In their model, users manually control the mouse's movement. The simulation concludes once the mouse either encounters an obstacle or reaches its destination [1].

Our goal is to enhance this model. We aspire to design an autonomous Micromouse maze simulation that leverages the Flood Fill Algorithm to determine the shortest path. Mirsha et al. and Tjiharjadi et al. have illustrated how to integrate the Flood Fill Algorithm with physical components to navigate a real Micromouse maze [3] [4] [5]. Adapting their work, we will focus solely on the algorithm, setting aside the hardware considerations.

Enhancements to Our Research

The Flood Fill Algorithm currently stands as the most prominent method for Micromouse competitions. Consequently, from an algorithmic standpoint, there's limited scope for refining our approach. However, a significant aspect we're choosing not to address in our research pertains to the diagonal movements of the mouse in real-life Micromouse contests. Our simulation restricts the mouse to move only forward, left, or right. A potential avenue for augmenting our research would be to modify both our simulation and algorithm to incorporate this diagonal movement capability. Nevertheless, for reasons of time constraints and simplicity, we have opted not to delve into this dimension for now.

References

[1] Martin Komák and Elena Pivarčiová. "Creating a Simulation Environment for the Micromouse". English. In: *TEM Journal* 11.1 (Feb. 2022). Copyright - © 2022. This work is published under https://creativecommons.org/licenses/by-nc-nd/4.0/ (the "License"). Notwithstanding the ProQuest Terms and Conditions, you may use this content in accordance with the terms of the License; Last updated - 2022-09-06, pp. 479–483. URL: http://login.ezproxy.lib.vt.edu/login?url=https://www.proquest.com/scholarly-journals/creating-simulation-environment-micromouse/docview/2702222085/se-2.

[2] *Micromouse robot runs maze in record-breaking five seconds (w/ Video)*. Image source. 20120. URL: https://phys.org/news/2010-11-micromouse-robot-maze-record-breaking-seconds.html.

[3] Swati Mishra and Pankaj Bande. "Advanced Algorithms for Micro Mouse Maze Solving". In: *Proceedings of the 2009 International Conference on Embedded Systems & Applications, ESA 2009, July 13-16, 2009, Las Vegas Nevada, USA*. Ed. by Hamid R. Arabnia and Ashu M. G. Solo. CSREA Press, 2009, pp. 78–84.

[4] Swati Mishra and Pankaj Bande. "Maze Solving Algorithms for Micro Mouse". In: *2008 IEEE International Conference on Signal Image Technology and Internet Based Systems*. 2008, pp. 86–93. DOI: 10.1109/SITIS.2008.104.

[5] Semuil Tjiharjadi, Marvin Wijaya, and Erwin Setiawan. "Optimization Maze Robot Using A* and Flood Fill Algorithm". In: *International Journal of Mechanical Engineering and Robotics Research* 6 (Sept. 2017), pp. 366–372. DOI: 10.18178/ijmerr.6.5.366-372.

[6] A. Tucker. *Applied Combinatorics, 6th Edition*. Online access: Center for Open Education Open Textbook Library. Wiley, 2012. ISBN: 9781118210116. URL: https://books.google.com/books?id=hdgbAAAAQBAJ.

[7] Bob White. *APEC MicroMouse Contest Rules*. 2004. URL: https://www.thierry-lequeu.fr/data/APEC/APEC_MicroMouse_Contest_Rules.html.