**Androids Dream of Electric Mice:**

**Navigating Mazes with Anki’s Cozmo**

Maxwell McMahon

Department of Computer Science

UNC Asheville

mmcmahon@unca.edu

**ABSTRACT**

Understanding and optimizing algorithms are cornerstones of computer science. Teaching these concepts can be a challenge for educators, due to the difficulty of creating effective visualizations. For Example, educators often demonstrate maze-solving algorithms via two-dimensional computer simulations, in which a disc navigates a maze. Using Anki’s Cozmo robot, we have developed software that navigates through a maze using pledge, random-mouse, and wall-follower algorithms. Using This software, we are able to visually complete a maze and compare the runtimes of each algorithm.

# INTRODUCTION

When discussing algorithms and robotics, one oft-mentioned example is maze-solving. To demonstrate these algorithms, educators often use animated simulations of mazes, in which a disc solves a maze step by step. Another alternative is having a robot demonstrate the maze. to accomplish this end, teachers often have their students create their own robots or use pre-made kits such as M-bot. However, Robot maker Anki has recently released their Cozmo robot. Cozmo is equipped with a front-facing camera, an accelerometer, and several sensors. Cozmo also is packaged with a python-based SDK. Due to Cozmo’s popularity, and powerful sensors/development tools it comes packaged with, we decided to use Cozmo to demonstrate several maze-solving algorithms. Using Cozmo’s SDK, we have created a framework for users to create and test their own algorithms. We also demonstrated Pledge, Random-Mouse, and Wall Follower algorithms. Using this framework, we ran Cozmo through a three-dimensional maze.

# BACKGROUND

While there are many different algorithms for solving mazes, we decided to test the Random Mouse, Wall Follower, and Pledge algorithms based on two different criteria. Any algorithm selected must be able to solve an unknown maze, algorithms such a dead-end filling require a top-down view of the maze, and are not able to solve a maze from within. In order to provide a stable framework for external users, we chose algorithms that would not require us to add more functions to our maze and navigation classes. Algorithms such as Tremaux’s algorithm and Fraenkel’s algorithm [1], while efficient, require the addition of functions for marking paths, determining whether a node is a junction, and determining the entry-path into a node.

# IMPLEMENTATION

## Structure

In order to navigate different mazes, we generated a two-dimensional graph. the graph consists of RxC nodes where R and C are the number of rows and columns chosen by the user. an exterior “frame” of nodes is also added, indicating nodes that are outside of the maze. Each node stores its coordinates, whether or not it is an exterior piece. Each node also links to its up, left, down, and right neighbors.

The graph is accessed and modified by a navigation class, which keeps track of the robot’s orientation and position within the maze. This navigation class also severs node connections within the graph whenever walls are discovered.

Within Cozmo’s drive program, we added a drive functions class, containing functions for advancing through the maze, rotating 90 degrees clockwise/counterclockwise, and sensing walls. Each function calls its counterparts in the navigation class, accessing and modifying the graph as more of the maze is explored

The physical maze consists of foam board walls, 3D printed corner pieces, and a foam board floor. each wall has a marker attached on both sides, allowing the robot to detect the wall with its frontal camera. When assembling the maze, walls are slotted into the edge pieces, and attached with double-sided tape to the board.

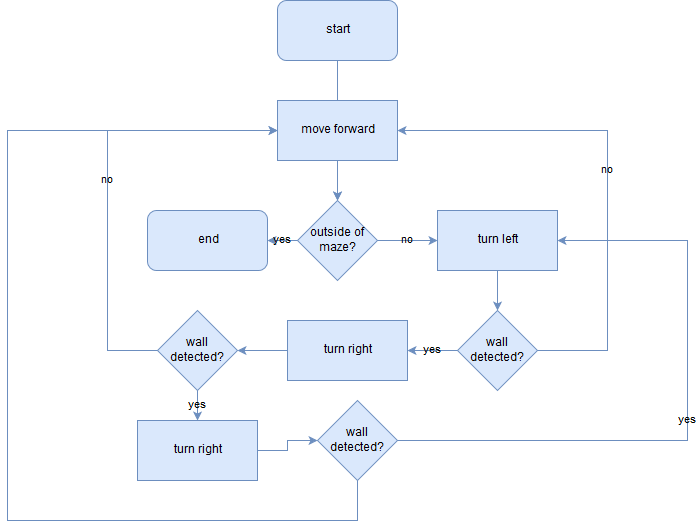
## Algorithms

* + 1. *Random Mouse*

The random-mouse follows a simple premise: after entering a node, choose a random path and advance along it. If the robot enters a dead-end, reverse and exit the node. Continue this process until the maze is exited.

Unlike most maze-solving robots, Cozmo can only sense objects in front of it, rather than all directions. To account for this, after choosing a random direction, we have Cozmo physically turn in that direction so that the robot can detect whether there is a wall in that direction. If there is Cozmo rotates in the reverse direction and chooses another direction. To improve efficiency, and to prevent the possibility of endlessly turning towards walled off directions, we added flags to each direction. Whenever a wall is encountered after choosing a direction, that directions flag is activated, and the robot will not turn in that direction again. If the flag for both the left, right, and center directions are activated, the robot has hit a dead-end and will reverse. When the current node is left, the flags are reset.

* + 1. *Wall-Follower*

The wall-follower algorithm has a left-hand and right hand version. Based on the version, the algorithm follows the right-hand or left-hand rule [2]. If there is not a wall on the rule’s hand, turn in that direction, otherwise continue until a wall is encountered in front of the robot, turning in the direction reverse of the rule. The wall-follower function is only applicable to mazes that simply connected [2]. (All walls are connected to maze border or each other). For our program, we used the left-hand version of the wall-follower algorithm. 

When the robot enters a node, it turns left to see if there is a wall to its left (no side sensors) if there is turn right and check for a wall, if a wall is detected, keep turning right and checking for walls until no wall is detected. when there is no wall in front of the robot, move forward and repeat.

* + 1. *Pledge*

According to Abelson and diSessa, this algorithm was developed by British student John Pledge at age 12 [3]. The Pledge algorithm solves one of the key issues with the Wall Follower algorithm: the requirement that the maze is simply connected. The algorithm starts by moving forward until a wall is encountered. the robot then sets its heading and sum of turns to 0. After the wall is encountered, the robot turns right. Check for a wall, and if a wall is found, keep turning right until a path is found, and advance. after this initial movement, the robot enters a loop. at each step, the robot will turn left and see if there is a wall, and if detected, turn right until a path is found, advancing after finding a viable path. After each left turn, we increase the heading by 90 degrees and sum of turns by 1, and after each right turn, we reduce the heading by 90 and number of turns by 1. This loop will continue until both heading and the sum of turns are equal to 0, after which the robot will continue advancing until another obstacle is encountered. (whenever the heading is equal to 360 or -360, it is reset to zero.) This algorithm is extensively proven by Abelson and diSessa [3].

## Test Plan

In order to test the maze, we constructed a random maze, and ran each algorithm on the resulting maze. In the case of all algorithms other than the random-mouse algorithm, we solved the maze by hand according to each algorithm, comparing the path the robot followed to the path generated by the algorithm. Due to the random nature of the random-mouse algorithm, the algorithm was considered successful is the robot was able to successfully exit the maze.

# CONCLUSION

As a beta software, usage of Cozmo’s SDK is not widespread. By creating our framework in a relatively new software, we help create a baseline for other members of the Cozmo SDK to create their own algorithms, or use the software as an example in other projects. As the SDK is updated and moves closer to release our framework can be improved with new features, perhaps even being modified by the community. By creating demonstrations of maze-solving algorithms we add a visualization that may help others reach a better understanding of fundamental computer science concepts.

# ACKNOWLEDGMENTS

We would like to thank Dr. Marietta Cameron, Dr. Kenneth Bogert, Dr. Adam Whitley, Hann Henson, and Max Dennis for their assistance and input on this project.

# REFERENCES

[1] Rao, Nagewara SV, et al. *Robot navigation in unknown terrains: Introductory survey of non-heuristic algorithms*. Technical Report ORNL/TM-12410, Oak Ridge National Laboratory, 1993.

[2] Babula, Michał. "Simulated maze solving algorithms through unknown mazes." *Organizing and Program Committee* 13 (2009).

[3] Harold Abelson, *Turtle Geometry: The Computer as a Medium for Exploring Mathematics*, mit pbk. ed., The Mit Press Series in Artificial Intelligence (Cambridge, Mass.: MIT Press, 1986, ©1980), 176-79, 191-98.