MicroMouse Maze Runner

Final Report

Team number: sddec20-28
Client: Dr. Phillip Jones
Advisor(s): Dr. Phillip Jones

Team Members
Joshua Wooi
Austin Chesmore
Aaron Walter
Richard Anderson
Tyler Fuchs
Jorge Gomez

Team Email
sddec20-28@iastate.edu

Team Website
sddec20-28.sd.ece.iastate.edu

Revised: 2020-11-15

Table of Contents

Figures & Tables

. Introduction

1.1 Acknowledgement

This section acknowledges Dr. Philip Jones for his contributions in providing technical advice to the team and Lee Harker for his guidance in sourcing hardware parts and using the EAGLE software. In addition, we would like to thank Jeremy Thurlby and his workshop who helped create feedback for the physical maze design and cutting the maze tiles.

1.2 Problem & Project Statement

Problem Statement:
Our client has outlined a robot that will be able to navigate a maze and autonomously decide and calculate the fastest path to the goal of the maze. This mouse must be able to navigate the maze and run the maze with the fastest path available. This mouse will need to demonstrate how the interconnection of hardware and software can function together to accomplish a task.

Solution Approach:
A MicroMouse Maze Runner Showcase.

People project that there will be many more computer/electrical/software-based jobs than there are people to fill the jobs, so we need to try to give a reason for future ISU students to consider robotics-based curriculums. This project serves to showcase the capabilities of engineering students from multiple disciplines at Iowa State University.

The expected outcome of this project is a functioning micromouse unit(s), capable of both autonomous and/or remote-control functions, that can traverse through several different maze designs effectively. Our expected outcome was near our completed design, but there were some major missed goals of our project. The main goal that we were not able to accomplish was the traversal of the maze autonomously. While our micromouse can navigate the maze with controller input, we were not able to finish the autonomous version of the maze navigation. We also did not finish a graphical user interface. The client outlined a GUI to draw the maze and update in real time, but we never finished this feature.

1.3 Operational Environment

The product of this project was to be able to operate a micromouse in a controlled environment. We designed the maze to have a smooth surface with adjustable walls. This will ensure the maze is stable for the micromouse to navigate. The operation of a micromouse outside of this environment condition is unexpected and undesired. We expect the environment to be modular and transportation friendly so people can use it anywhere with enough space.

1.4 Requirements

1.4.1 Engineering Constraints

- The memory size on the Feather was a strict 520 kb

- The ROM size of the Feather is 4MB
- The micromouse requires a flat smooth surface for the maze
- Hardware is inherently unreliable

1.4.2 Functional Requirements

Functional requirements of this project would be that the mouse is able to move through the maze, is able to map a maze to a maze model, is able to find the shortest path of the maze model and is able to follow paths through the maze. We also must have a maze that can fit the micro mouse, be modular, transportable, and solvable by the mouse.

1.4.3 Non-Functional Requirements

Non-Functional requirements of the project are that the mouse should be accurate, reliable, and sturdy. Our maze will have to be of quality materials so that it is able to last long when we reconfigure or transport it.

1.5 Intended Users & Uses

As a showcase project, the intended user of a micromouse is anyone with latent interests in Robotics. Our team designed this project to inspire and encourage anyone to pursue a degree in the STEM field.

1.6 Expected End Product & Deliverables

During this project there were many timelines that COVI-19 forced us to change due to circumstances we had no control over. Therefore, some deliverables were either delayed indefinitely or removed entirely from the scope.

Components ordering (2/21):
We submitted our Bill of Materials for ETG to approve, so we can continue to make progress on Prototype 1.

Delay (3/20)
When classes transitioned into a virtual environment, we had to delay the deliverables with Prototype 1 and all future hardware related operations until further notice, since the university had implemented safety precautions against COVID by closing access to the lab equipment on campus to students.

GUI (3/27)
The team created and delivered the first prototype of the GUI to the client.

Virtual Maze (3/27)

The team developed a virtual maze and robot with the unity engine so work could continue the prototype's embedded algorithm while conforming with virtual instruction guidelines.

GUI 2 Electric Boogaloo (4/10)
The team designed the GUI to be more user-friendly and included a mapped maze.

Hardware Constructed (4/18)
The team constructed the parts to ensure everything was functioning as expected.

Second Round of Parts (9/15)
The team ordered a second round of parts as a contingency for technical failures during tests.

Software Testing (9/29)
The team began software testing on the Feather so that we can continue to develop hardware and software integration.

Maze cut & Assembled (10/10)
The team cut the physical maze and assembled it to test the micromouse once we completed the prototypes.

PCB Design (10/27)
The team ordered, received, and assembled our first prototype on a working PCB.

Prototype Assembly (11/6)
The team delivered a working prototype that someone can control with an Xbox One controller to the client.

1.7 Relation to Products

We expect that a competition-ready micromouse will have various iterations before it is up to professional standards. The team believes that there is still room for improvements on our first prototype, and it is therefore unfit for a micromouse contest. That said, we believe that this prototype has met all the requirements and expectations of our client's project. A micromouse should not be more than 25cm by 25cm, in length and width, with no restrictions on the height of the micromouse. Also, it must use a source of energy that does not involve a combustion process. Next, the micromouse should not leave behind parts of itself in the maze while solving it. Lastly, micromice should not jump, fly, or climb over any walls, and it should not burn, cut, or damage the walls of the maze or maze itself. This mouse is 10.2cm x 10.2cm which is smaller than most other micromouse units. Furthermore, a lithium-ion 3.7V battery powers the mouse.

2. Specifications & Analysis

2.1 Proposed Approach

The team decided to prototype the project using an Adafruit Feather as a controller, to allow for a more robust implementation using the Feather's features. The team also chose the Feather since it can support all the peripherals for the project, such as I2C signal bus, and analog to digital signal conversions. The feather also has built in support for Bluetooth and Wi-Fi functionalities that satisfies the requirements of our client for an optional wireless communication system with a user.

We had two prototyping phases for hardware in this project. The first phase was assembling components on a breadboard, and the second phase was designing and having a PCB fabricated. The plan when starting the project was to use Python to interface the software and hardware, due to the flexibility of the language in its compatibility with other types of software. For instance, we would write the maze algorithm, the GUI, and hardware interface in Python with different libraries. Additionally, we utilize C++ along with several Arduino libraries to program the functionalities of the micromouse.

We ordered an additional set of components, so that we would have spares if potential errors or hardware malfunctions occur.

2.2 Design Analysis

During the first semester, the project team thoroughly researched the specifications that the Micromouse project will need to meet. Through this research, the team developed a components list. We brought the component list for approval by Dr. Jones and ETG, to ensure that the selected components were reasonable for the purposes of the project. In the second semester, the team focused on testing the components and integrating them into a full micromouse prototype. At the end of the second phase of the design process the team transitioned into testing integration of software and hardware to ensure functionality. At this point, the team decided to split into two branches for software and hardware. Towards the end of the project, the team converged both branches back together to reintegrate the software and hardware side of the project.

2.3 Conceptual Sketch

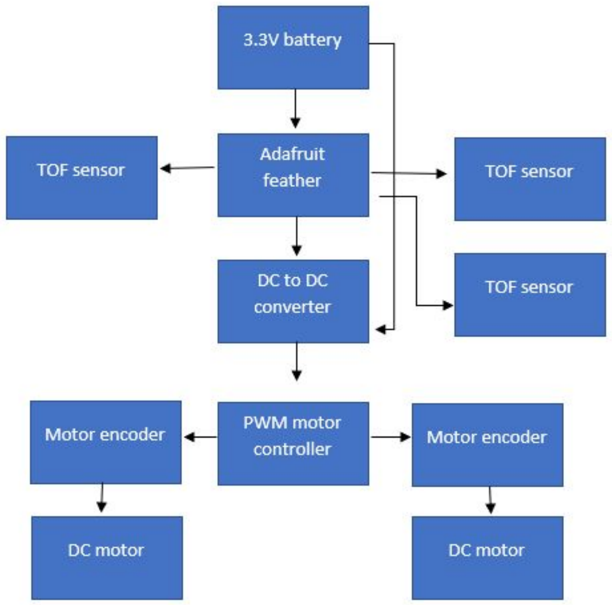


Figure 2.1 High-Level Schematic Design of MicroMouse Prototype

This is a conceptual design of how the micromouse will be created and interact with all the components that were selected by the tram for the project. It shows how the micromouse functionality will live on the Adafruit feather, while the feather will then command the other components to enable the micromice hardware functionality

3. Implementation Details

3.1 Previous Work & Literature

Classwork here at Iowa State University has laid a foundation that we can reference, specifically CprE 288 in its autonomous robot project. This project has had a long history, and there are plenty of online articles and documentation that will help us in designing the micromouse. There are also micromouse competitions that should yield even more documentation that our team can reference for our project. For our PCB design we utilized a software program called Eagle and referenced a tutorial by Jim Blom (Blom, Jim).

3.2 Technology Considerations

The technology that we incorporated for our micromouse was cost effective and modular. The team selected widely available components to integrate it into the prototype design. By using off the shelf components, the project will less likely have quality issues, such as solder bridging or improper design. This removes several potential failure points, allowing for the project to be bug tested more effectively. However, one weakness of using these types of components is that there is less controllability of the specification.

Other considerations include the amount of memory available to our team on the feather. We need to be diligent on how large our software is and how much memory it will consume when running the maze.

3.3 Possible Risks & Risk Management

Prototyping the mouse was the most worrisome of our project, getting all the different pieces to work together included. After we learned what hardware can work with what software, prototyping became much easier and we were able to create more designs.

When we tested components, it was important to be careful as to not connect pins where they should not be. This became apparent when a team member put a 12V supply on a GPIO pin and subsequently fried our first Feather. Other risks include damage to our components, when moving them they needed to be either secured in a safe storage container, or back in the foam protection they came packaged with. Luckily, we did not have any issues with bent pins from transportation.

Cost was a major part of our potential risk when designing our micromouse robot. The cost to the maze can range from 10s to 100s of dollars. We only ended up spending about $40 on the maze itself but looking back we should have spent more.

The final risk that we considered was that none of our members of our team had any prior experience with PCB design. This was why our team spent many weeks designing the PCB. To mitigate this risk, we had two team members, with instructions from our mentor, build and practice with Eagle software to become more efficient.

3.4 Expected Results & Validation

The desired outcome, as outlined by the client, is a micromouse that can solve and run through a maze on its own. It will also have a display that people will use to run the program and can respond to user input. The user will have the ability to control the mouse with a controller and the mouse will respond accordingly.

4. Timeline, Resources, & Challenges

4.1 Project Timeline

Dates	Deliverables
1/20-2/21	We constructed our groups and scheduled bi-weekly meetings with the clients. The team researched parts for the robot prototype and ordered them. The team began prototyping.
2/22-3/20	The university transitioned to online formats and this included our team. We developed a virtual platform for our micromouse to navigate a maze and allowed us to test any maze path finding.
3/21-4/10	Prototypes of the GUI, maze and Virtual maze delivered to the client. The team decided to move work to a virtual platform built in Unity due to COVID-19.
4/11-4/18	Our team received our parts after COVID-19 delayed them. Hardware testing began with breadboard and components.
4/19-5/2	Project Design Document drafted and finished; Project website updated.
Break	Summer Break
8/20-8/31	The team scheduled meetings with the client.
9/1-9/14	Our team Encoders tested. Round 2 of parts ordered.
9/15-9/28	GUI connecting via Wi-Fi, microcontroller code continuing. PCB design beginning. Hardware testing continued. Round 2 of parts received.
9/29-10/9	A* algorithm developed. Updated version of GUI developed. Web server on feather developed and tested.
10/10-10/26	Maze cut and assembled. Final GUI developed to incorporate a maze drawing panel. First version of PCB design and presented. A* finalized. Modified flood fill development.
10-27 - 11/9	PCB delivered. Prototype 1,2, and 3 fabricated. Input to micromouse demonstrated to the client. Modified flood fill finalized.
11/10-11/19	Prototype 1 & 2 fully functional. Project Document drafted and finished. Build Guide created for future teams.

Table 4.1 Deliverables Schedule

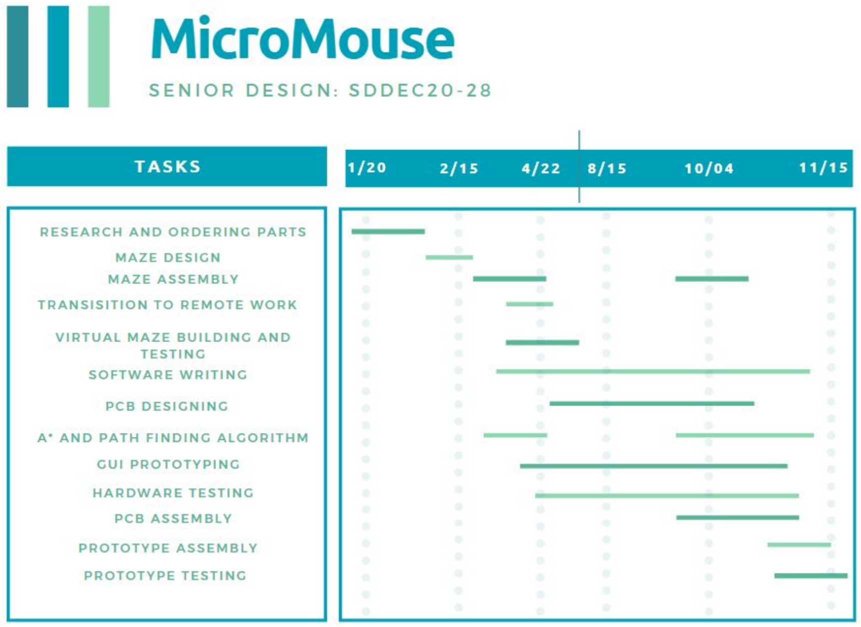


Figure 4.1 Gantt Chart showing timeline

4.2 Feasibility Assessment

- Autonomous maze navigation
    - Wall detection without getting stuck in a loop
    - Detecting the goal of the maze
- User input controls
    - Bluetooth protection from other devices
- Maze Design
    - Price - keeping the price in check so we do not blow the budget.
    - Mobility - being able to transport the maze while also being large enough to be complex

4.3 Team Member Contributions

Team Member	**Contribution**	**Prev semester**	**This semester**	**Total**
Richard Anderson	My contributions were the algorithms for the micromouse, specifically the modified flood fill and A* algorithms, as well as testing the algorithms correctness and modifiability to for hardware use	47	75	122
Austin Chesmore	My contributions to the project we to primarily assist with testing the hardware components. I also assisted with assembling/soldering of the prototypes and the component selection	53	105	158
Tyler Fuchs	My main contribution to the group was working on the UI in its many forms and building the physical maze. In total I built 4 different UIs to test and two sets of maze walls.	42	91	133
Jorge Gomez	My greatest contribution to the team was the scouting of hardware and PCB research and design.	41	74	115
Aaron Walter	My contribution was writing the firmware for the feather. I got a mini web server working on the Feather and all of the "drivers" written to work with the hardware. I also helped a lot with the PCB designs.	57	117	174
Joshua Wooi	My contributions were designing and revising the PCBs and some of the soldering work.	37	70	107

Table 4.2 Personal Effort Requirements

4.4 Other Resource Requirements

The main outside concern is where we can store the maze while it is being built. As per the client, they would like a maze that is at least 9 feet by 9 feet, so this will require a large open space.

4.5 Financial Requirements

For the Micromouse project our budget is set at $500 per semester with the ability to extend beyond if we need to. The team split the budget between the hardware and physical components. We decided to construct the maze from rough cut pine to keep the price down. In total the physical maze ran a price tag of $32.15. Whereas someone can use the hardware between different prototypes. During the first half of the project the team spent $316.87 on the hardware components. On the other hand, we used $320.42 for the second set of components. In the end the sum cost of all hardware came to a cost of $605.14. As depicted in Figure 4.2, one of the greatest costs is the ToF sensors. We found a different component for future use. The suggestion is to go from using the VL6180 ToF sensors by Adafruit and moving towards using the VL680X ToF sensor from Pololu causing the price of sensors to go from $13.95 per sensor to $8.49 per sensor. Overall, the team kept costs the allowed amount and new, cheaper components are being thought of for future use.

Item#	Item type	Qty	Cost	Subtotal	Part#	Supplier	Supplier#
1	Feather board	4	$20.95	$83.80	3591	Digi-Key	1528-2514-ND
2	Motor Controller	4	$3.42	$13.68	713	Digi-Key	2183-713-ND
3	Magnetic Encoder	8	$7.95	$63.60	4761	Digi-Key	2183-4761-ND
4	Wheels	8	$3.75	$30.00	1452	Digi-Key	2183-1452-ND
5	Motors	8	$18.95	$151.60	3053	Digi-Key	2183-3053-ND
6	DC/DC 3.7 to 12v	4	$2.99	$11.96		Canton Electronics	
7	ToF Sensors	12	$13.95	$167.40	3316	Digi-Key	1528-1815-ND
8	Battery 3.7v (4.2)	4	$9.95	$39.80	PRT-13813	Digi-Key	1568-1492-ND
9	Encoder Wires	8	$0.95	$7.60	4762	Digi-Key	2183-4762-ND
10	.3.6-pin 0.1" Short Break-away Male Header - Pack of 10	2	4.95	$9.90	3009	adafruit	
11	36-pin 0.1" Short Female Header - Pack of 5	2	7.95	$15.90	3008	adafruit	
12	0.1" 36-pin Strip Right-Angle Female/Socket Header (5 pack)	2	4.95	$9.90	1542	adafruit	
	TOTAL	62		$605.14			

Figure 4.2 Final Build of Materials and Related Cost

5 Testing and implementation

5.1 Hardware and software description/build

Testing of the hardware components and assembling several testing prototypes was necessary to ensure that the teams ordered components would function correctly. At an early point in the project the team moved towards creating a virtual maze environment that team members could test the micromouse in. During this time, another team member was constructing the physical maze. However, the team ended up testing the first hardware prototype with limited functionality due to component shipping wait times, and thus the team scrapped the virtual maze for testing and transitioned into the physical maze.

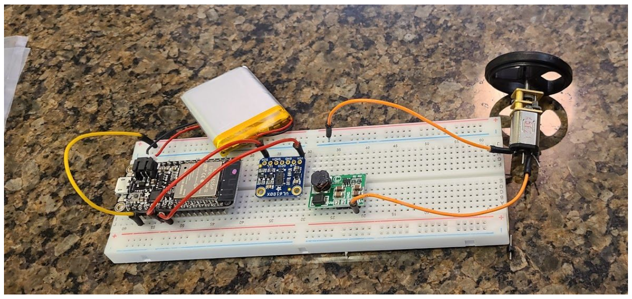


Figure 5.1 First Prototype Build

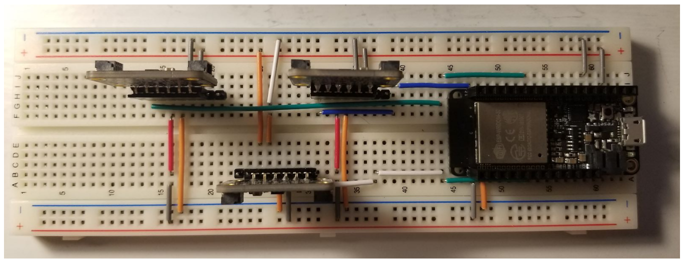


Figure 5.2 Breadboard Prototype Build

This hardware prototype at the end of semester one used a breadboard and the available components, for basic proof of concept design tests. We only incorporated a single wheel and sensor into the design to demonstrate that all components work together as intended. Additionally, after the first round of testing, the team would then move to a more permanent hardware design using PCB instead of the breadboard.

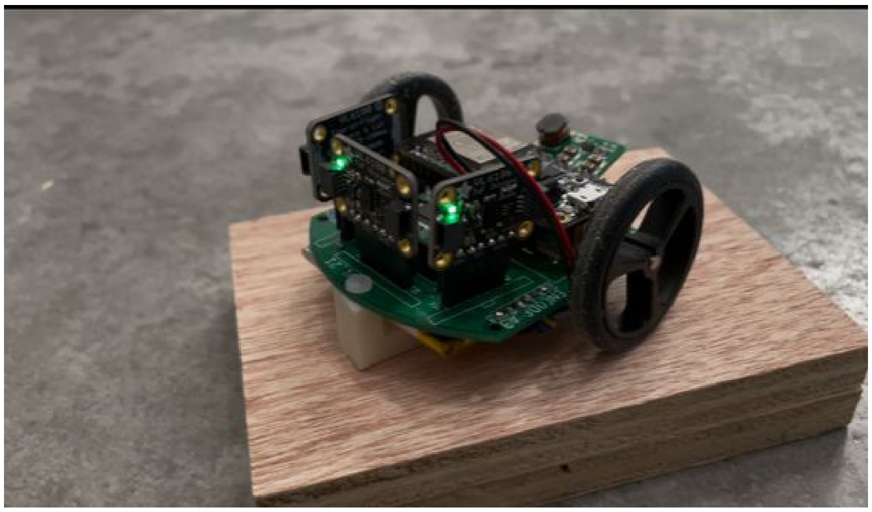


Figure 5.3 Final Prototype Design

The current micro mouse design is currently being prototyped using a final PCB design for proof-of-concept design tests. The team integrated all components onto the PCB along with a 3D printed ski. For this prototype we were able to utilize the C++ software that was written by the team to achieve movement functionality.

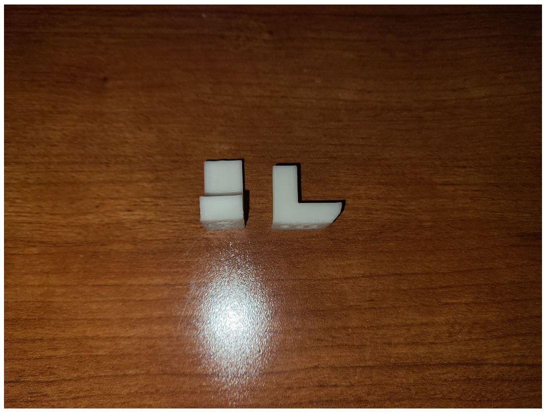


Figure 5.4 Micromouse Ski

Additionally, the project team 3D printed a ski that is seen implemented in figure 5.2. This ski allowed the prototype design to function with only 2 wheels, while also allowing for improved stability.

5.2 Hardware Details

The team selected the components members could easily integrate them into the micromouse. By using prebuilt components, we reduced the risk of having issues with the quality of the components. This would allow for the team to focus on the main objectives, rather than having to troubleshoot smaller issues such as faulty components.

Below, you will find a breakdown of the components used in the construction of the micromouse along with the quantity of parts used per micromouse, and the reasoning behind why the team chose them.

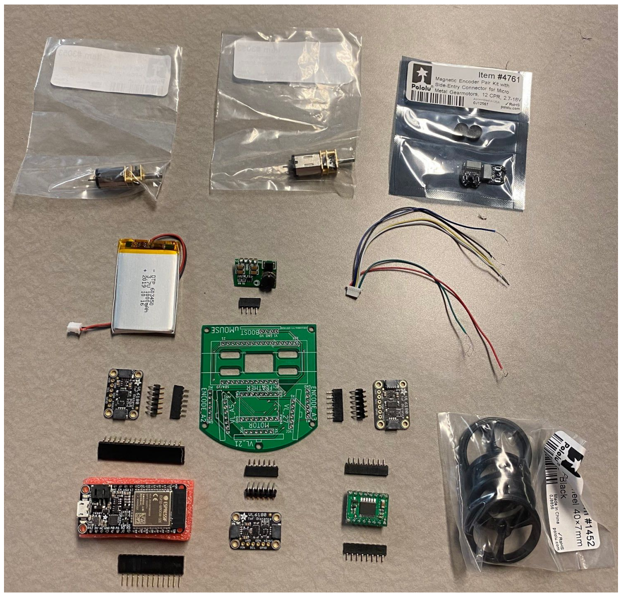


Figure 5.5 Micromouse Components

- HUZZAH32 ESP32 FEATHER MALE HDR …………………………...……………...**x1**
    - The Adafruit Feather is a compact but powerful development board with in-built functionalities that meet all our requirements. It has Wi-Fi capabilities, programmable in C/C++, uses I^{2}C for the sensors, PWM for the motors, ADC/DAC pins for controls, and GPIO pins for everything else.
- TB6612FNG Dual Motor Driver ……...…………………………………..…………….**x1**
    - This Motor Driver can control up to two motors simultaneously and allows us to tune the rpm of the motors as needed, using feedback from the encoders.
- Magnetic Encoder Pair Kit With Side-Entry Connector for Micro Metal Gearmotors, 12 CPR, 2.7-18V …………………..……………………………………………………..**x1**
    - The encoders allow for precise rpm configuration and are compatible with the motor in the components list.
- WHEEL 40X7MM PAIR - BLACK ………...…………………………………………….**x2**

    - These wheels are compatible with the motors on this list. In addition, they are rubber lined and will do well on different surfaces. Lastly, the axles are made of plastic to eliminate any possible problems with the encoders.

- GEARMOTOR 220 RPM 12V METAL EXTD ……………………………………**x2**
    - These motors meet our size requirements, are powerful and fast for their size, include gearboxes, are compatible with a wide range of wheels and the encoders.
- 12W DC-DC Converter Boost 2V-12V …………...…………………………………...**x1**
    - This boost converter increases and regulates the voltage output of the battery to supply power to the motors.
- VL6180 TIME OF FLIGHT DISTANCE
…...…………………………………………...**x3**
    - These distance sensors use I^{2}C for fast communication with the Feather and fit well within the size for the micromouse.
- B6-PIN FEMALE JST SH-STYLE CABLEATTERY LITHIUM 3.7V 1AH …..……...**x1**
    - This battery meets our needs to power the Feather, at 1000 mAh, is rechargeable, and is relatively compact.
- 6-PIN Female JST SH-STYLE Cable
    - These six pin male connectors fit well with the encoders.
- 36-pin 0.1" Short Break-away Male Header (10pack) ...………...……………………...**x1**
    - These male header pins are to be soldered onto components so it can be attached to header sockets on the PCB.
- 36-pin 0.1" Short Female Header (5 pack) ……………...……………………………....**x1**
    - These female header pins are to be soldered onto the PCB so it can accept components.
- 0.1" 36-pin Strip Right-Angle Female/Socket Header (5 pack) ………………………...**x1**
    - These right angle male pins are to be used to allow the VL6180 Time of Flight Distance sensors to stand perpendicular to the PCB.
- PCB ……………………………………………………………………………………...**x1**
    - The PCB was the only piece of hardware that was designed by the team for the explicit purpose of housing/organizing the above components. In doing so, it gives the micromouse a more cohesive look. We decided on using header pins/sockets style

connections for a secure but impermanent connection between the PCB and the components. This was done so components can be easily replaced. This PCB was designed using Eagle, a scriptable electronic design automation application, and went through a few iterations before arriving at the final version that was used. More information on the PCB can be found in Appendix II.

5.3 Software Details

The Micro-Mouse project has multiple software components. There are three main parts, the GUI, the software to bridge the hardware devices, and the maze algorithms. Each component was essential in delivering a maze running micro-mouse that met all of our client's requirements.

The device firmware that interfaces with the hardware components was written in C++. Many different functions were needed that were not originally foreseen when beginning the prototyping phase. For instance, a function was required to slow down the acceleration of the motor controller due to a bug that would shut down the feather if the speed was set from a stop to full speed.

The User Interface has gone through many different changes and versions. To begin the project, we had selected a GUI library in python called PyQT5 to write the graphics. The plan was to write a simple panel with a spot for a console, image holder for mapping the maze, and a few buttons for functionality. After a few versions and upgrades to this plan we ran into an issue with drawing the maze and updating it during the navigation of the maze. Instead, our group moved to developing a web-based GUI that will utilize JavaScript to run commands and access the basic web server running on the Adafruit Feather. HTML5 would provide the imaging mapper tag that would help to draw the maze and fill in the missing functionality that the PyQt library was missing.

The maze solve algorithm is used so that the mouse can navigate and then solve the maze. The algorithm first does a modified flood fill to build a maze model, and then uses an A* algorithm to find the shortest path. The algorithm is gone into further detail in section 5.3.2 and 5.3.3.

5.3.1 Device Firmware

The device firmware has many different components that make the device function. It implements a lightweight MVC framework written in C++.

Access Point
The first component is the AP Station software that allows the ESP32 to host a wireless hotspot for the "command center" client to connect to. The micromouse firmware can also be used to connect to a client, to then connect to an existing hotspot if needed.

Async Web Server (Controllers and Views)

The firmware also includes an asynchronous web server to allow the command center to communicate with the micromouse in real time. There are two parts to this web server: the JSON REST API and a WebSocket. The REST API allows the client to query the status of the robot: distance traveled, sensor distances, speed, and other real time information. The WebSocket allows the command center to send movement commands to the robot. The valid inputs are currently keyboard and any X-input controller.

Components (Models)
The firmware includes a lightweight "driver" for each of the devices connected to the mouse. There are components set up for the encoders, motors, and time of flight sensors. These components control the logic of the hardware and provide information to the pathfinding algorithm.

5.3.2 Flood Fill

The mouse uses a modified flood fill algorithm to scan the physical maze. How a typical flood fill algorithm works is it scans all possible directions from the current cell, and then recurses down each of the cells that are able to be visited, until every cell has been visited. This is the behavior of the typical flood fill, whose pseudocode can be found in Appendix IV.

In terms of the micromouse however, this algorithm is not satisfactory, since the mouse needs to physically move throughout the maze. With the described algorithm, the mouse would be expected to teleport back to previous cells, as well as already knowing the maze. We needed to modify this algorithm so that the wall-checking instead uses scanner distances to check for immediate walls, as well as consider mouse movement and directions, and moving the mouse to previous cells during recursion. In our implementation, we create a 2d array of binary numbers, and use the 5 rightmost bits to keep track of the north, east, south, and west walls, as well as if the related cell has been visited.

The updated flood fill starts with the mouse using its sensors to scan the immediate left, right, and forward cells from its current direction, and then adding those scan results to the maze map. From those scans, if any of them are 0 (meaning no wall), then the mouse will turn towards the "wall-less" direction (if need be), move forward 1 cell, and then recurse down in the algorithm with that cell. Once the cell has gone down each direction, the mouse turns towards the direction opposite of what the mouse was at the start of the current recursion level, then moves forward, effectively moving the mouse backwards to where it was before recursion. The cell is also marked as being visited so the mouse will not go back down the direction. The modified version of the flood fill can be found in Appendix IV

5.3.3 A*

Once the mouse has been flood filled, it can use an A* algorithm to find the shortest path from the starting position to the goal.

To find the goal of the maze, we obtain the map through the flood fill process. Then, we have another function which searches the maze map for a 2x2 closed loop, meaning we make sure that a northwest cell can go to the south and east cells, and the southeast cell can go to the north and west cells. This functionality can be seen in Appendix IV.

Once we know the end goal, we have the heuristic to be used for our A*. From there, it's a matter of getting a selected path, taken during the A* process. With this path, we will be able to get instructions that the micromouse can follow to solve the maze.

5.4 Maze Design

The client outlined that the micromouse maze needed to be mobile and reconfigurable. To combat these challenges our team designed a maze that used wood tiles/panels in combination with a peg board pattern where the walls could be moved around to create new maze patterns. This is shown in appendix II. Figure 5.6 is an image of the maze we built when testing our prototype micromouse. We did not use the maze panels for this test because we wanted a complex design for the robot to navigate through.

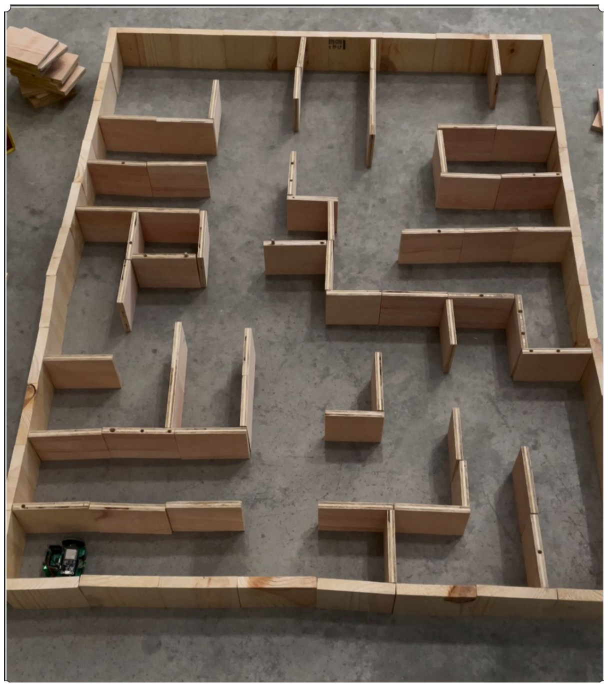


Figure 5.6 Physical Maze Assembled for Prototype Testing

These maze walls are the standard 3 inches high. This was found to be the standard for half size micro-mice. For the maze panels, they were cut from a 12” x 8’ board of rough-cut pine. Since it was a rough cut board we were forced to cut a little further down and the panels came out to be 11.25” x 11.25”. This required the panels to precisely cut and therefore we spoke to Jeremy Thurslby, and with his help we cut the panels using a CNC router which yields perfectly symmetrical panels that can be connected together.

5.5 PCB Design

5.5.1 Circuit Schematic

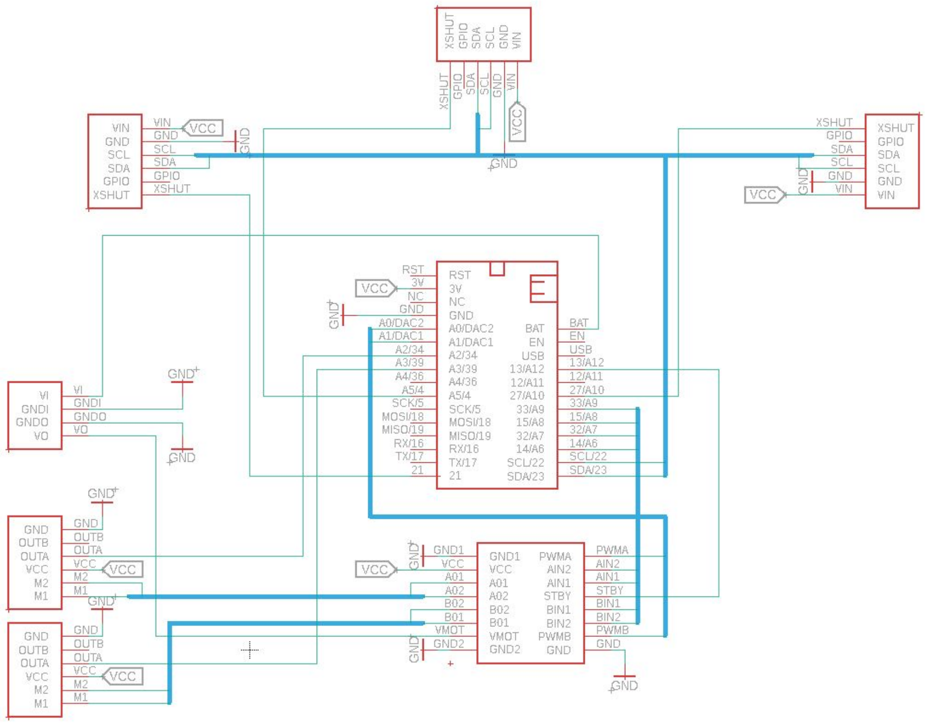


Figure 5.7 Circuit Schematic for prototype.

The circuit schematic file (depicted above) illustrated through Eagle, which describes a low-level interpretation of our initial design, and details the interconnections between our components in the PCB. The feather board serves as our main controller for most of the operations through the BAT, 3V, and most of the GPIO pins. The 3 distance sensors sit at the front/top and communicate with our Feather board via I2C in the SCL and SDA pins. Additionally, we allocated 3 GPIO pins as ‘XSHUT’, which will effectively shut down a targeted sensor, should we decide it is necessary. The 3V, which outputs a steady 3.3V through an in-built regulator, serves as the pull-up supply for most of the components. However, this pin is limited to a 500mAH output, which will not be sufficient to power the motors. The BAT pin, which outputs a 3.7V supply, is therefore boosted to a steady 12V to power the motors, through the motor controller. Most of the other GPIO pins are dedicated to the ADC and PWM pins which decide the rpm at which the motors operate. Feedback from the motor, provided by the encoders, is then connected to the Feather. All these components are externally powered by a 3.7V, 1000mAH rechargeable battery, which is rechargeable through the microUSB port on the Feather.

5.5.2 PCB

Rather than using any surface mounted pads, we created custom footprints to solely use through-hole vias to accommodate header pins. Using Eagle's auto-routing function, we produced a layout (depicted below) that we used for our MicroMouse prototype. The produced layout had multiple vias that connected traces between the 2 copper layers, and while we had reservations about this design from an aesthetic standpoint, we do not expect that it will hinder performance in the slightest.

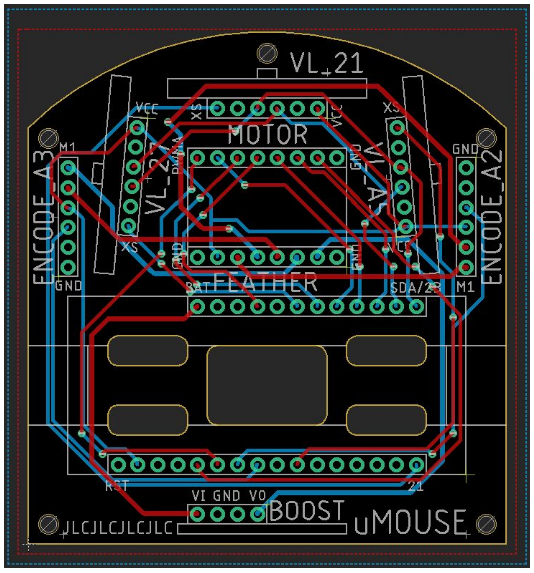


Figure 5.8 PCB Version 2

5.6 Testing Process

Once the software was built and flashed onto the micromouse we were able to test the basic web server that was hosted on the Adafruit Feather. An interface between the software and hardware was tested by flashing this firmware onto the mouse and seeing any changes in the robot's responses. This development was all done in CLion as described in Appendix I. We also tested some of the hardware controllers with Google Test, which is a c++ unit testing library.

5.7 Results

Throughout the duration of this project our team has put in countless hours from design planning to PCB drafting, to sawing the maze, to draft writing. Our client outlined a few main goals for us to work towards along with a project statement. The main goals we had were to:

1. Build a working micromouse that can manually navigate a maze.
2. Build a User interface so any person can test the functionality of the micromouse
3. Allow the robot to navigate the maze autonomously and choose the fastest route.

Along with these outlined criteria the team established success criteria for the team to judge the outcome of the project. While our team's timeline and results were affected by the COVID-19 pandemic, the team persevered and met the majority of the self-imposed goals. Below is a list of the success criteria that the team created, along with the results and explanations for meeting or not meeting criteria.

1. **Create a working micromouse design that has easily replaceable components**

The results of the success criteria were that we successfully created a couple of fully operational micromouse prototypes that had easily replaceable components. this was accomplished by utilizing male and female header pins to solder to the PCB and components, as opposed to soldering all components directly to the PCB

2. **Produce a micromouse prototype that utilizes a PCB**

The result of the success criteria was positive as we received our components and started prototyping originally with breadboards. However, breadboard testing created several reliability issues with faulty connections. these “bad connections" prevented the components from communication effectively over I2C. This pushed the team to get a PCB design created and ordered. after the PCB prototypes were integrated into the prototype design the micromouse functioned much more reliably before.

3. **Allow for micromouse to have a manual user operational mode**

The result of this success criteria was successful. The web server that is hosted on the Feather allows the user to send commands from controller input, and the mouse will respond accordingly at a rate of 50 commands per second. The number of commands was necessary for the controller input to accurately reflect the changes in the input. In addition to the web server, an acceleration class was created to deal with a bug with the motor controllers. If the encoders were set to 1,1 (left, right) from 0,0 the program would crash, and the Feather would need to be reset. An acceleration class handled the input from the user interface and slowly scaled the speed the motor controllers received. For example, if a command of 1,1 was sent to the Feather over the web server, the firmware would set the motor controllers to 0.2, 0.2 and after 200 milliseconds would increase by increments of 0.2 until the speed matched the input.

4. **Use a flood fill based algorithm to run through the entire maze while mapping all the walls and the end goal.**

The results of this success criteria are a mix of success and failure. We have a working modified flood fill algorithm that can search through a virtual maze, and create a map based on it using theoretical sensor input, mouse movement, and mouse turning.

Unfortunately, we were not able to get the flood fill working with the physical mouse and physical maze in time, but we believe the algorithm could be modified or reused with functions that physically move the mouse wheel or use the mouse's sensor output.

5. **Use an A* algorithm to find the shortest path between the starting position and the end goal.**

The result of this success criteria is also a mix of success and failure. We do have a working A* algorithm that takes the generated maze map from flood fill and generates the shortest path from the start to the goal. The output of our A* is the instructions the mouse will need to execute to reach the end goal of the maze.

Unfortunately, the mouse does not currently use this list of instructions, but the functionality is there for when a later group is able to make the mouse execute the instructions one by one.

6. **Create a GUI interface that will allow for a user to easily interact with the micromouse platform**

The outcome of this success criteria is also a mix of success and failure. The team made a move late in the development process to switch the python user interface to a web-based. The transition was made to allow more team members to contribute to the development of the user interface. Some of the functionality that was required by the client was completed. The web-based UI displays information to the user that the client outlined, along with input being received from a controller input. The functionality that was missing was mapping the maze to the interface. This functionality was also missing from the Feather, so implementation was not completed on either front.

7. **Micromouse prototype should comply with full size compilation standards.**

When it comes to micromice, there are two options the team explored, full size or half sized. Due to the size of the parts the team selected, we decided on going with the full size bot. The team attempted to create a mouse as small and compact as the full size would allow, leading to nearly being considered half sized mouse. The standards for a full size include:

- Self-Sustaining - Should not be controlled or use an energy source that involves a combustion process.
- Structurally Sound - Should not leave any parts behind while traversing the maze.
- Movement Restrictions - Should not jump, fly, or climb over, as well as damage, the maze walls or floor.

- Size - Should not be more than 25 centimeters in width or length, but there is no restriction to the height, as stated by the IEEE SAC 2018 micromouse competition rules.

The current design of the micromouse has met these expectations except the rule regarding not controlling the micromouse, however this was due to our client requesting the user be able to manually control the mouse. No current issues with its structural integrity. Lastly, the micromouse measures 10.2cm x 10.2cm.

8. Create a portable maze that will be solved by the micromouse.

The result of this success criteria is also a mix of success and failure. We currently have a bunch of timber that we are able to form in the shape of a maze, but the dimensions and distances between cells aren't consistent. Currently the software expects consistent spacing between cells and between walls. The maze was only developed to hold a small maze, a total size of 33" x 33" with the panel system. There were some other issues like miscommunications of how wide the cells should be, but we were able to get past these problems by changing how we constructed the maze. To work around this problem the maze was constructed without the use of the tiles and allowed the size of the maze to increase by a factor of 3. Future teams will need to develop a better tiling system if they continue with this type of approach.

6. References

1. Blom, Jim. “Using Eagle: Schematic.” Using EAGLE: Schematic,

2. learn.sparkfun.com/tutorials/using-eagle-schematic/all.

3. Hmcgrath, and Instructables. “The Basics of SolidWorks: a How-To Guide.” *Instructables*, Instructables, 20 Oct. 2017, www.instructables.com/The-Basics-of-SolidWorks-A-How-To-Guide/.

4. J. Cai, X. Wan, M. Huo and J. Wu, "An Algorithm of Micromouse Maze Solving," *2010 10th IEEE International Conference on Computer and Information Technology*, Bradford, 2010, pp. 1995-2000, doi: 10.1109/CIT.2010.337.

5. https://www.youtube.com/watch?v=1AXwjZoyNno&ab_channel=JeremyBlum

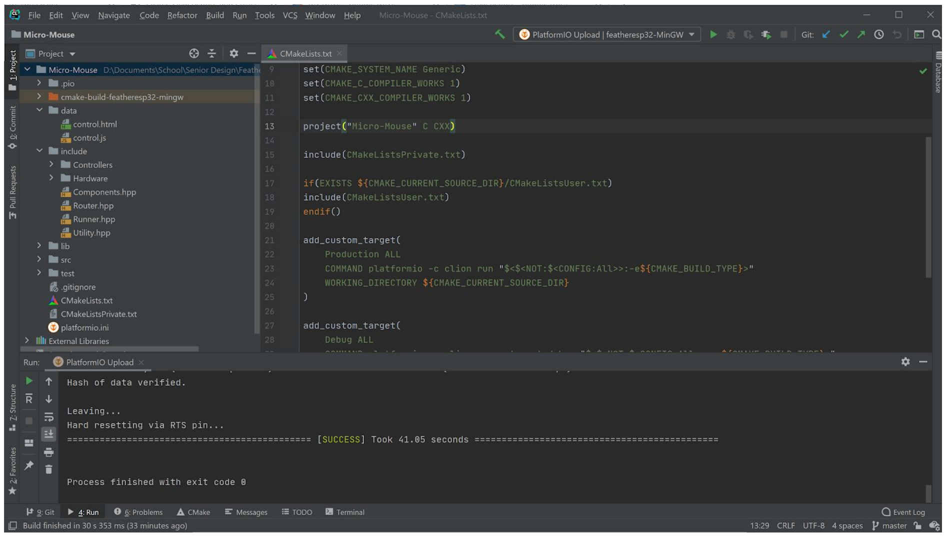
6. Data sheet. . Retrieved from https://www.pololu.com/file/0J86/TB6612FNG.pdf

Appendix I

Operation Manual

To begin using the micromouse, code will need to be flashed onto the Adafruit Feather from a computer. Our team used the ide CLion to write and compile our software interface that was written in C++. In addition to CLion, MinGW with Platformio will also need to be installed to build and flash our program onto the Feather. Mingw is the software toolchain that will compile the software using GCC.

A comprehensive build guide can be found on our website so that future teams assigned to this project will have a visual guide on project set-up and configuration. This guide will include downloading all the necessary components and each set of software. The image below is a screenshot of CLion and the building platformio building section, used to flash code onto the micromouse. On successful flashing of the Feather, you will receive a success message as seen in the console in the below screenshot.



```
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_C_COMPILER_WORKS 1)
set(CMAKE_CXX_COMPILER_WORKS 1)

project("Micro-Mouse" C CXX)

include(CMakeListsPrivate.txt)

if(EXISTS ${CMAKE_CURRENT_SOURCE_DIR}/CMakeListsUser.txt)
include(CMakeListsUser.txt)
endif()

add_custom_target(
    Production ALL
    COMMAND platformio -c clion run "$<$<NOT:$<CONFIG:All>>:-e${CMAKE_BUILD_TYPE}>"
    WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
)

add_custom_target(
    Debug ALL
```

```
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
============================================= [SUCCESS] Took 41.05 seconds =============================================


Process finished with exit code 0
```

Build finished in 30 s 353 ms (33 minutes ago)

Figure A.1 CLion

Our current controls for the micromouse are from an Xbox controller with tank drive input. Tank drive means that each analog stick on the controller responds to one wheel of the robot. These settings were used to test new code on the feather along with console outputs on the web ui.

After flashing the feather from CLion the web server should be running so the user needs to connect to the hotspot on the micromouse. This enables the user to run the web UI and successfully connect to micromouse via the websocket. Note that when connected to the micromouse no internet will be available.

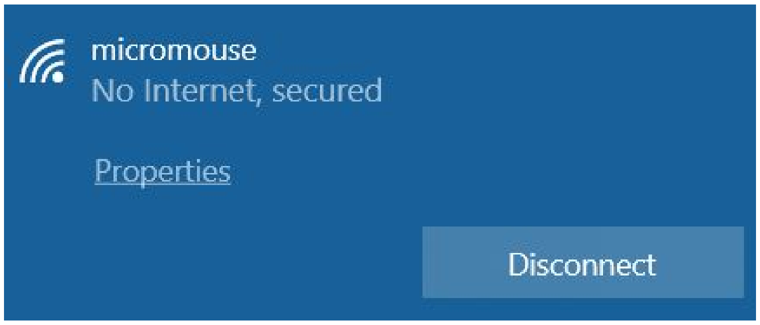

micromouse
No Internet, secured

Figure A.2 Micromouse Connection

Since this device cannot access the internet, multiple workstations are recommended when testing. When our team was testing, we had two members, one connected to the micromouse and the other writing and testing code and flashing it to the micromouse. Since any device can flash the project onto Feather, we found this to be the fastest way to make changes and run these changes. The Adafruit Feather has a light that would turn red to display an input that it was receiving, this was necessary for our group to test because we could use this display to see if the controller was giving the robot commands. This was useful during the testing phase when the controller would lose connection to the computer. We were able to see that the robot was still grabbing commands even after the disconnection, and to solve this problem we added a kill command if the controller input was lost during a run.

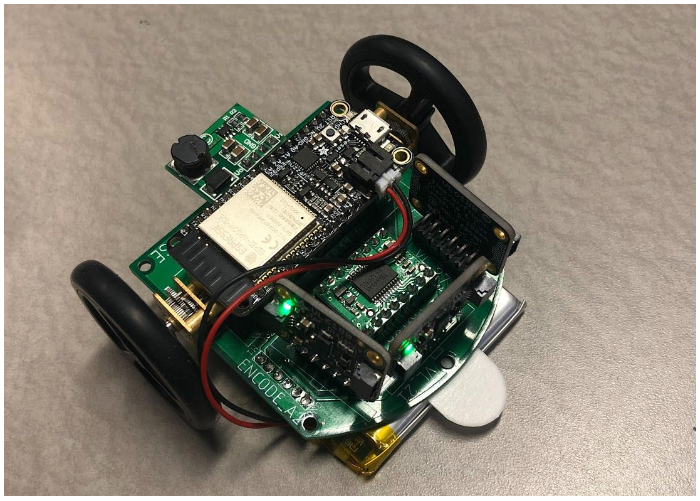


Figure A.3 Micromouse Prototype

Appendix II

Alternative/Initial Version

During the design of the micromouse many changes were made with the goal of optimizing the performance of the project while still sticking to the guidelines for a micromouse. Below is a list of initial or altered parts of the micromouse.

- Virtual Micromouse
- PCB
  - At the current moment, the PCB has reached Version 4.
    - Version 1 was designed with the idea of being able to place every component on the board while staying within the restriction of a micromouse size. Five holes were made on the board. The largest of the five holes is placed right above the mouse's encoders. This hole will allow for encoder wires to attach to the encoder above the board and latch on to the PCB underneath. The four other holes are placed to help with keeping the motors in place. In addition, for this version, Eagle's autorouting feature was used.
    - Version 2 saw the addition of five holes, one at each corner and the fifth at the very front of the board. These holes are added in case mounting components was a desire for the user. For example, the hole at the very front of the board will be used to attach a ski meant to help the mouse run smoother. In addition to this change, the board now has new traces. These traces are easier to follow, but they are still autotraced.
    - Version 3 has manually routed connections. This was done to create more coherent traces that are easier to follow and takes the shortest path between terminals possible. Improvements in performance because of these connections is minimal when compared to previous designs. Feather was turned 180 degrees on board as well to allow for more routing room.
    - Version 4 has new footprints for the updated ToF sensors that are more easily available. This decision also made it so that all needed components may be sourced from one place.

User Interface

The Graphical User interface (GUI) was the aspect of our project that changed the most. When the project began last semester, Python was chosen to display information to the user along with debugging information that would be useful when our prototype was completed. The first version of the UI (user interface) was simple and had no functionality, however later versions became more robust.

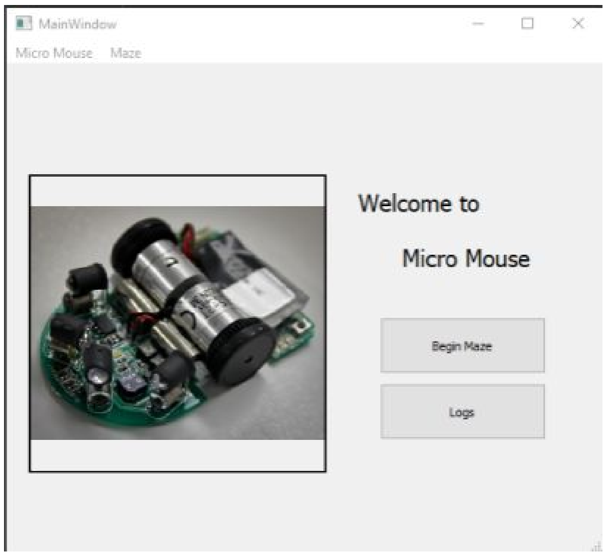
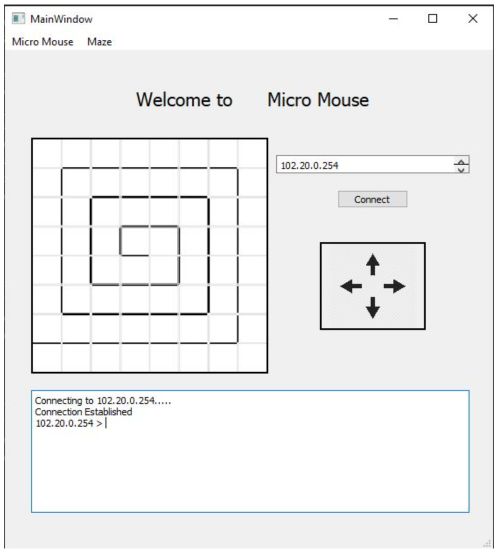


Figure A.4 UI version 1. Simple layout

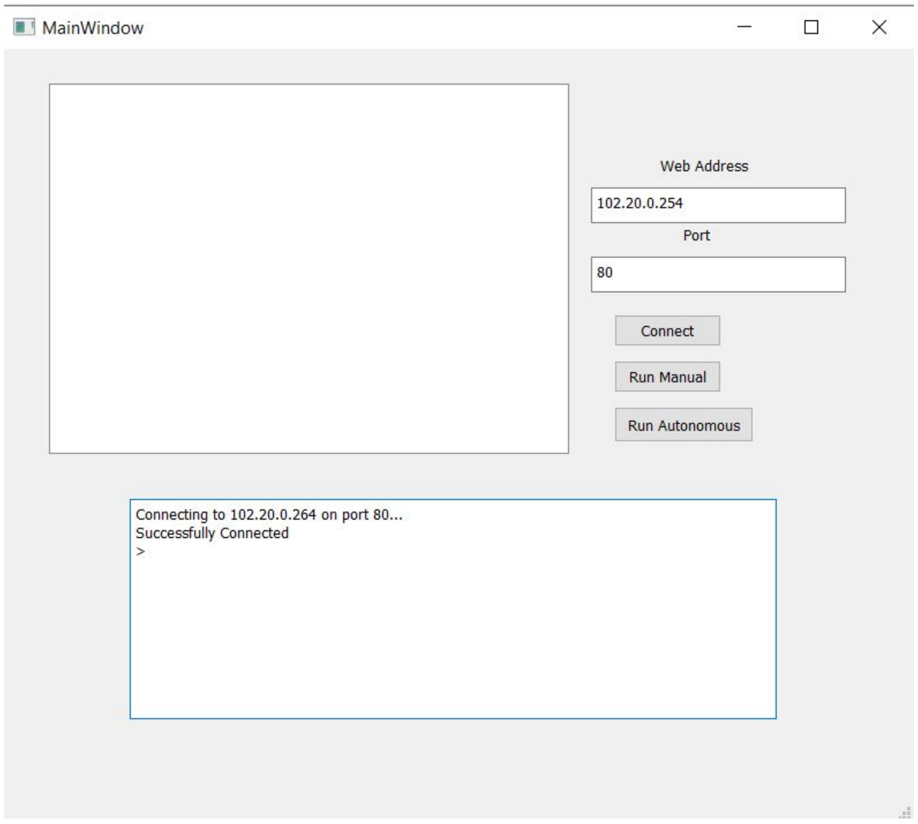
This template was shown to the group as a test version of what functionalities were needed and where we could see it being used. This version had a separate window for the actual maze, but nothing was displayed as we didn't have any of the infrastructure in place to make it usable. When version 2 was being designed we had a better idea of how things were going to be displayed and what was needed. At this point, the web server on the Adafruit Feather was being constructed so a text box was added to allow the user to pass the address of the feather and connect to it. On successful connection a message was displayed to the console as well.



```
Connecting to 102.20.0.254.....
Connection Established
102.20.0.254 > |
```

Figure A.5UI version 2 of the prototype. Web address added

Version 2 of the UI was a big step up as we could now connect to the web server that hosted the Feather. Version 3 was produced as the UI that would be used to test the functionality of the micromouse itself. This version was developed to add functionality of a port to connect to, along with buttons to select between the autonomous and manual mode. These modes would run the robot accordingly when the functionality is added to the micromouse itself. In the end, we did not use this GUI as we implemented a web-based UI.



102.20.0.254

80

```
Connecting to 102.20.0.264 on port 80...
Successfully Connected
>
```

Figure A.6 Version 3 of the UI. Added connectivity changes and manual/autonomous mode

MAZE

The maze was divided into two parts, the panels and the walls. The walls had to be modified slightly due to miscommunications with the actual size of the micromouse. But in the end, we had a mouse that could navigate the maze within the operational environment that we outlined.

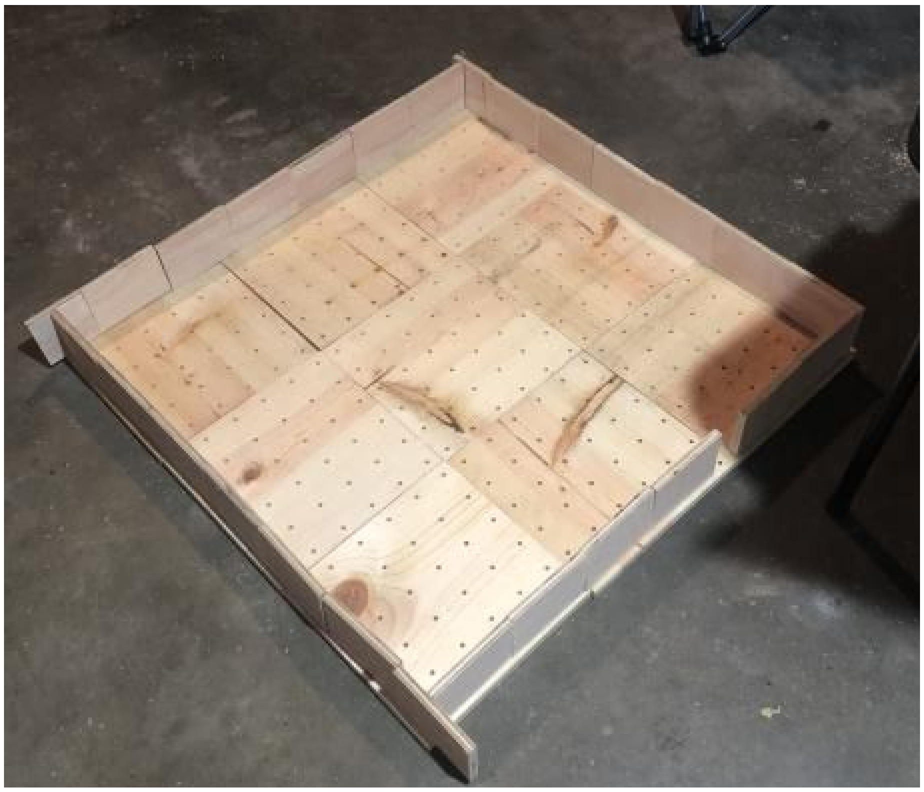


Figure A.7 Maze Panels

1.Appendix III

Tyler

For technical skills I have used python in previous classes for various things, but I had never tried to accomplish this type of functionality in python. Working with the PyQT package I was able to successfully build a GUI that could connect to the micromouse web server, however I was never able to finish the maze mapping to the screen. This was because PyQT did not have a library to draw a dynamic image and update throughout the maze run like I wanted it to. Another reason I was not able to finish this task was because we made the decision to move to a web-based UI so that other members on the team could contribute to the workload. This was not originally in the plan but I was in need of help to finish the UI so making this change seemed like it would have a greater impact on the team instead of myself struggling to figure it out. I was a little disappointed to see the work I had done in the past few months being deprecated, but as we looked into the future this was a necessary move for our team to make to continue work at the pace we needed.

One funny moment I would like to share is when we successfully were able to move the robot with the Xbox controller I finally received the satisfaction of the success our team was having in creating a maze running robot. Seeing the robot move successfully was a step along the way but I feel it was a key moment for our group to boast moral and continue working on our goal.

Jorge

During this project I have refined my ability to self teach in order to learn new material. Picking up a new skill is a difficult task but thankfully when I would get stuck I had the opportunity to bounce ideas around with my team and I found that to be something that I really enjoyed. For example, during this project I learned PCB design and whenever I would hit a wall I could depend on my teammates.

Austin

While working on this project I learned about how project planning is a crucial aspect of any successful project. I learned about the process of implementing hardware together with software in a team environment. Additionally, I learned about the difficulty that adapting to a chaotic world event can place on a project timeline.

While testing on the adafruit feather with the motor controllers and DC to DC boost the wiring became unstable and high voltage was connected to a GPIO pin. this resulted in that feather board being no longer functional. Fortunately the team has just received a new order of components and the team progress was not jeopardized.

Richard

While working on this project I got some good experience with c++, which I don't have a ton of experience with, as well as working with a team.

I also gained some good experience with learning how to deal with and work around requirements and limitations. For example, the micromouse only has so much memory that can be used, so the algorithms need to be done in a way that doesn't waste memory, and does not leak any memory.

Another example of a limitation is that the feather uses a specific c++ compiler with a shortened down version of the standard library, so I needed to make sure what I was making was able to compile on the feather as well as still work.

Aaron

During this project I learned how to design PCBs from scratch. I normally only write software, so it was interesting to learn how to create hardware. Because I worked on the PCBs, I also learned how to interface the software to the hardware. I ended up learning a lot about IoT devices and how they control hardware over the internet.

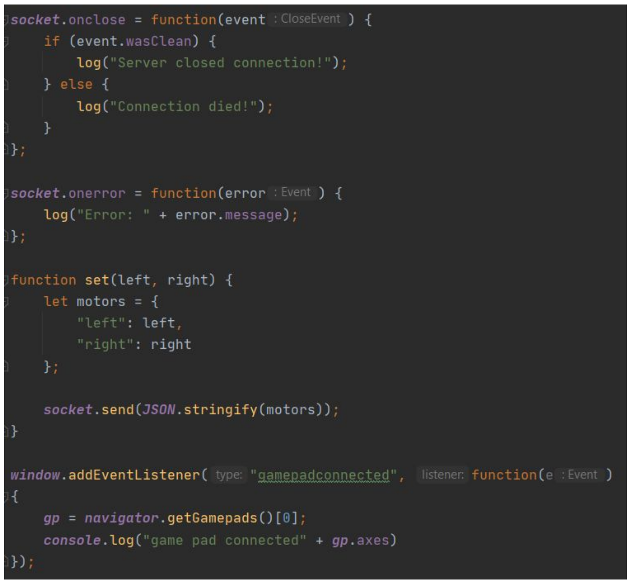
It was also interesting to learn how to program a microcontroller. Unlike traditional web applications, microcontrollers have a limited amount of memory and CPU power, making it easy to overwhelm the processor. Designing around this type of hardware was both challenging and fun.

Joshua

This project has been a rewarding experience. I gained new skills in PCB designing, and had the opportunity to practice my soldering skills. I am also grateful and honored to be able to work with the members of the team, whom I can confidently say will move on to be bright and competent engineers.

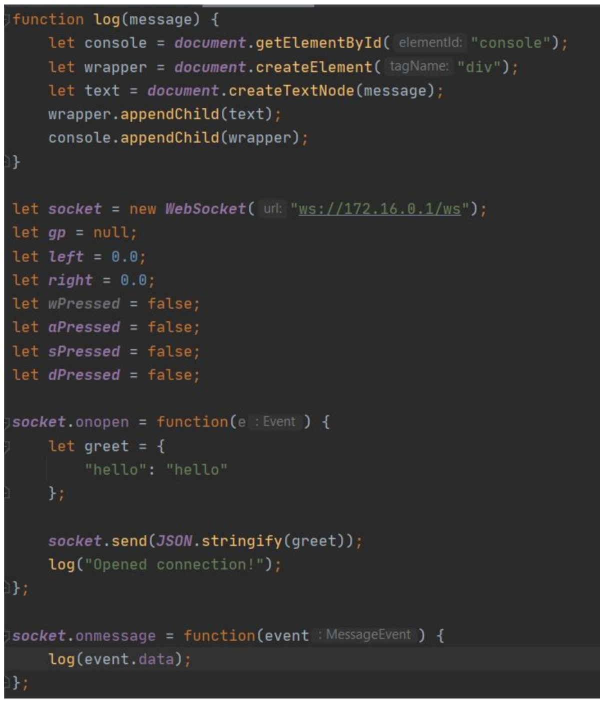
2.Appendix IV

The Javascript code that handles movement to the robot via the Xbox controller can be found below. This functionality was much easier to develop due to javascript libraries that were already available to us. A gamepad library was used that allowed us to easily develop and check for inputs on any connected game controller to the computer. This was necessary because the robot needed to be connected to the computer and since the computer could also connect to the controller by bluetooth we could send input from the bluetooth connected controller to our Wi-Fi connected robot. Since all xbox controllers output data the same we were able to test our design with an Xbox elite controller and a Xbox One controller without having to change any code.



```
socket.onclose = function(event : CloseEvent ) {
    if (event.wasClean) {
        log("Server closed connection!");
    } else {
        log("Connection died!");
    }
};

socket.onerror = function(error : Event ) {
    log("Error: " + error.message);
};

function set(left, right) {
    let motors = {
        "left": left,
        "right": right
    };

    socket.send(JSON.stringify(motors));
}

window.addEventListener( type: "gamepadconnected",  listener: function(e : Event )
{
    gp = navigator.getGamepads()[0];
    console.log("game pad connected" + gp.axes)
});
```

Figure A.8 Code Part 1



```
function log(message) {
    let console = document.getElementById( elementId: "console");
    let wrapper = document.createElement( tagName: "div");
    let text = document.createTextNode(message);
    wrapper.appendChild(text);
    console.appendChild(wrapper);
}

let socket = new WebSocket( url: "ws://172.16.0.1/ws");
let gp = null;
let left = 0.0;
let right = 0.0;
let wPressed = false;
let aPressed = false;
let sPressed = false;
let dPressed = false;

socket.onopen = function(e : Event ) {
    let greet = {
        "hello": "hello"
    };

    socket.send(JSON.stringify(greet));
    log("Opened connection!");
};

socket.onmessage = function(event : MessageEvent ) {
    log(event.data);
};
```

Figure A.9 Code Part 2

Figure A.10 Virtual Maze Design

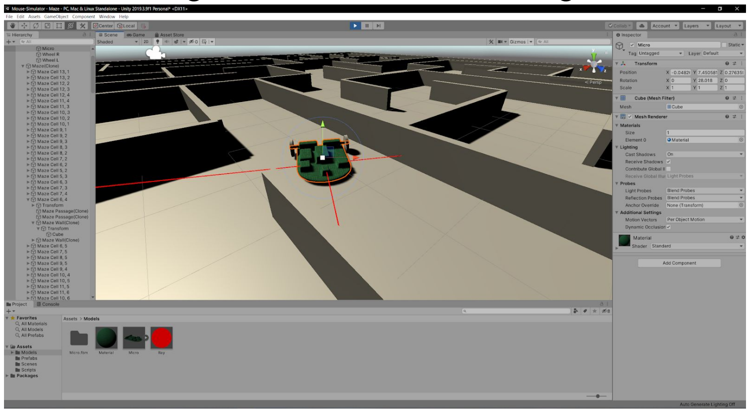
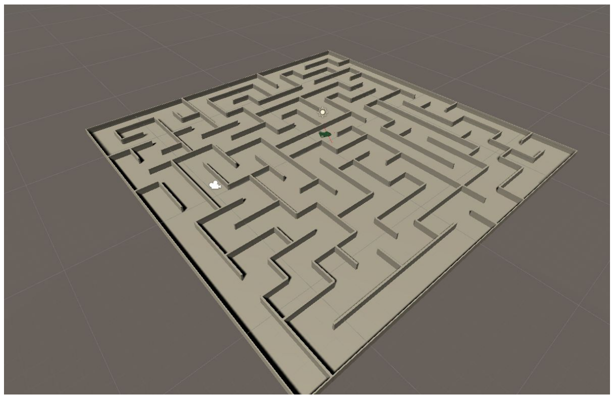


Figure A.11 Virtual Maze Layout



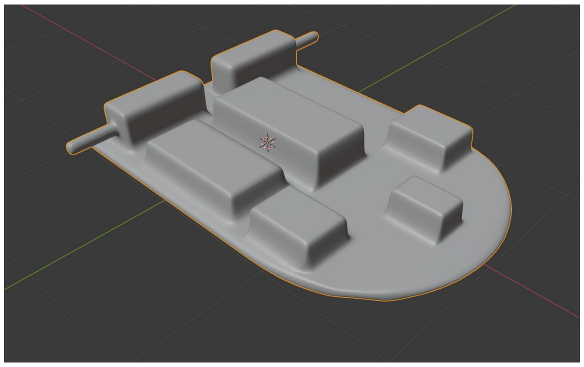


Figure A.12 Virtual Micromouse in Blender

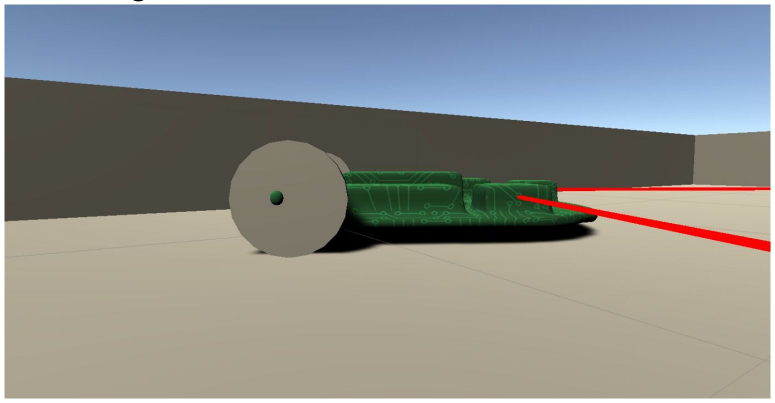
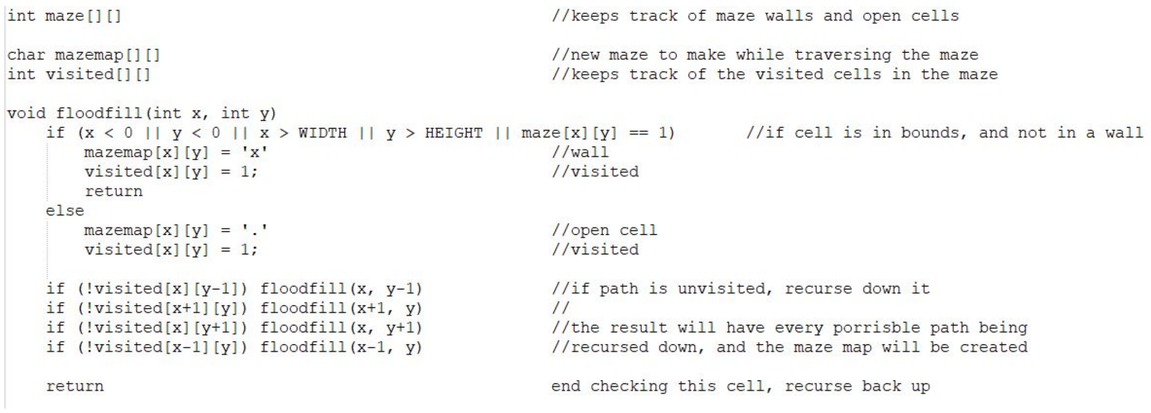
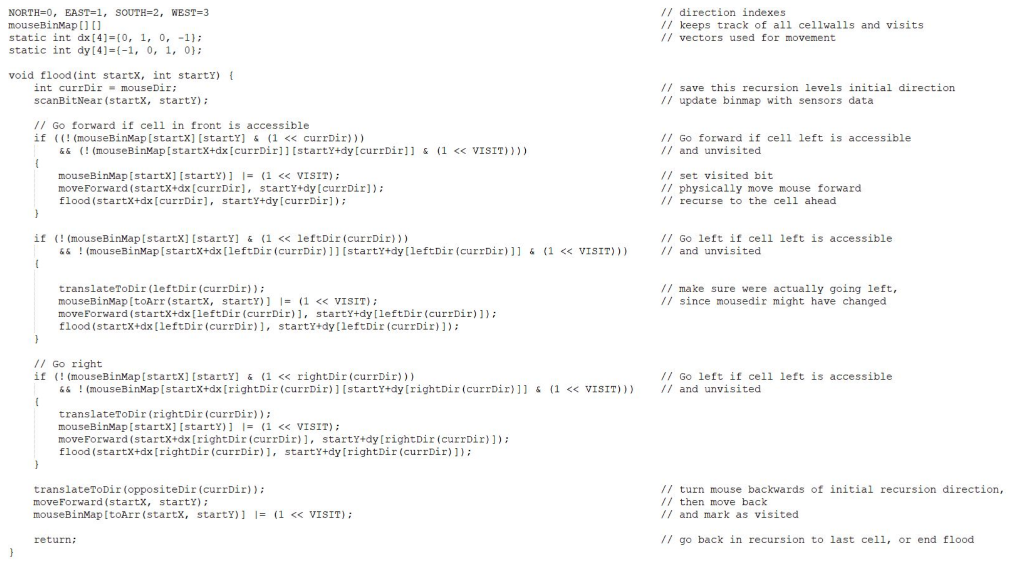


Figure A.13 Importing prototype into Unity



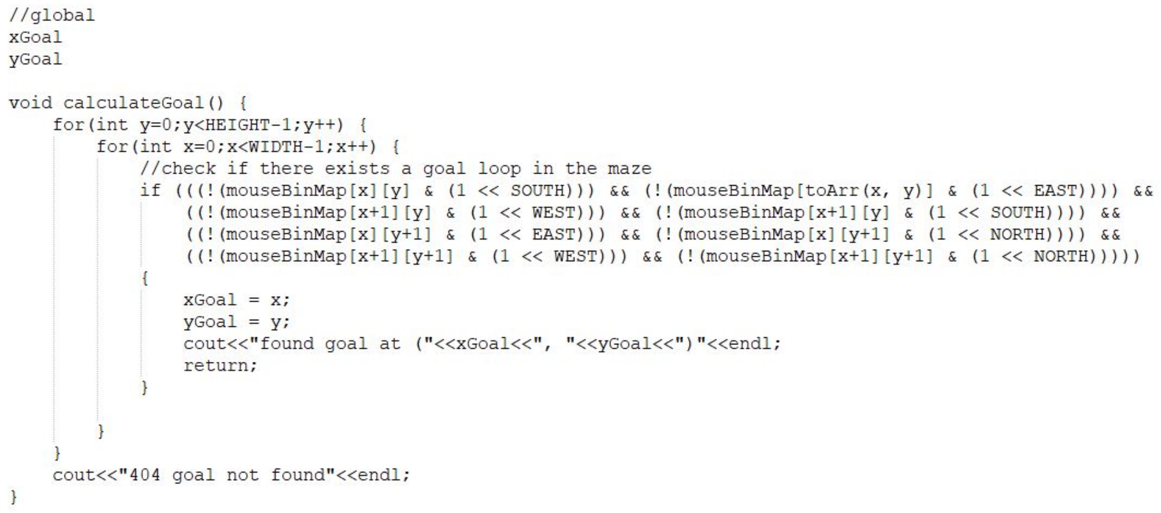
```
int maze[][]                                        //keeps track of maze walls and open cells

char mazemap[][]                                    //new maze to make while traversing the maze
int visited[][]                                     //keeps track of the visited cells in the maze

void floodfill(int x, int y)
    if (x < 0 || y < 0 || x > WIDTH || y > HEIGHT || maze[x][y] == 1)       //if cell is in bounds, and not in a wall
        mazemap[x][y] = 'x'                         //wall
        visited[x][y] = 1;                          //visited
        return
    else
        mazemap[x][y] = '.'                         //open cell
        visited[x][y] = 1;                          //visited

    if (!visited[x][y-1]) floodfill(x, y-1)         //if path is unvisited, recurse down it
    if (!visited[x+1][y]) floodfill(x+1, y)         //
    if (!visited[x][y+1]) floodfill(x, y+1)         //the result will have every porrisble path being
    if (!visited[x-1][y]) floodfill(x-1, y)         //recursed down, and the maze map will be created

    return                                          end checking this cell, recurse back up
```

Figure A.14 Standard flood fill algorithm (pseudocode)



```
NORTH=0, EAST=1, SOUTH=2, WEST=3                                                        // direction indexes
mouseBinMap[][]                                                                         // keeps track of all cellwalls and visits
static int dx[4]={0, 1, 0, -1};                                                         // vectors used for movement
static int dy[4]={-1, 0, 1, 0};

void flood(int startX, int startY) {
    int currDir = mouseDir;                                                             // save this recursion levels initial direction
    scanBitNear(startX, startY);                                                        // update binmap with sensors data

    // Go forward if cell in front is accessible
    if ((!(mouseBinMap[startX][startY] & (1 << currDir)))                               // Go forward if cell left is accessible
        && (!(mouseBinMap[startX+dx[currDir]][startY+dy[currDir]] & (1 << VISIT))))     // and unvisited
    {
        mouseBinMap[startX][startY)] |= (1 << VISIT);                                   // set visited bit
        moveForward(startX+dx[currDir], startY+dy[currDir]);                            // physically move mouse forward
        flood(startX+dx[currDir], startY+dy[currDir]);                                  // recurse to the cell ahead
    }

    if (!(mouseBinMap[startX][startY] & (1 << leftDir(currDir)))                        // Go left if cell left is accessible
        && !(mouseBinMap[startX+dx[leftDir(currDir)]][startY+dy[leftDir(currDir)]] & (1 << VISIT)))     // and unvisited
    {

        translateToDir(leftDir(currDir));                                               // make sure were actually going left,
        mouseBinMap[toArr(startX, startY)] |= (1 << VISIT);                             // since mousedir might have changed
        moveForward(startX+dx[leftDir(currDir)], startY+dy[leftDir(currDir)]);
        flood(startX+dx[leftDir(currDir)], startY+dy[leftDir(currDir)]);
    }

    // Go right
    if (!(mouseBinMap[startX][startY] & (1 << rightDir(currDir)))                       // Go left if cell left is accessible
        && !(mouseBinMap[startX+dx[rightDir(currDir)][startY+dy[rightDir(currDir)]] & (1 << VISIT)))    // and unvisited
    {
        translateToDir(rightDir(currDir));
        mouseBinMap[startX][startY)] |= (1 << VISIT);
        moveForward(startX+dx[rightDir(currDir)], startY+dy[rightDir(currDir)]);
        flood(startX+dx[rightDir(currDir)], startY+dy[rightDir(currDir)]);
    }

    translateToDir(oppositeDir(currDir));                                               // turn mouse backwards of initial recursion direction,
    moveForward(startX, startY);                                                        // then move back
    mouseBinMap[toArr(startX, startY)] |= (1 << VISIT);                                 // and mark as visited

    return;                                                                             // go back in recursion to last cell, or end flood
}
```

Figure A.15 Modified flood fill (partially pseudocode)



```
//global
xGoal
yGoal

void calculateGoal() {
    for(int y=0;y<HEIGHT-1;y++) {
        for(int x=0;x<WIDTH-1;x++) {
            //check if there exists a goal loop in the maze
            if (((!(mouseBinMap[x][y] & (1 << SOUTH))) && (!(mouseBinMap[toArr(x, y)] & (1 << EAST)))) &&
                ((!(mouseBinMap[x+1][y] & (1 << WEST))) && (!(mouseBinMap[x+1][y] & (1 << SOUTH)))) &&
                ((!(mouseBinMap[x][y+1] & (1 << EAST))) && (!(mouseBinMap[x][y+1] & (1 << NORTH)))) &&
                ((!(mouseBinMap[x+1][y+1] & (1 << WEST))) && (!(mouseBinMap[x+1][y+1] & (1 << NORTH)))))
            {
                xGoal = x;
                yGoal = y;
                cout<<"found goal at ("<<xGoal<<", "<<yGoal<<")"<<endl;
                return;
            }

        }
    }
    cout<<"404 goal not found"<<endl;
}
```

Figure A.16 goal searching (partially pseudocode)