# A-Maze-Ing Algorithms

**New Mexico Supercomputing Challenge Final Report**

**Team: 16**

**Cottonwood Classical and Del Norte High School**

**Team Members:**

Ayvree Urrea ayvreeurrea@gmail.com

Kiara Onomoto kiaraonomoto@gmail.com

Violet Kelly kellyviolet1111@gmail.com

**Sponsor Teacher:**

Karen Glennon

Patty Meyer

**Mentor:**

Flora Coleman

# Table of Contents:

# Executive Summary:

In psychological research, hierarchical mazes have been widely used to create and test the cognitive processes and functions of a human. Our interest in mazes was first sparked by an interest in the psychology field and human decision-making within different personality types. We wanted to analyze how humans made decisions in quick-paced situations that are presented to them on the spot. Mazes have this aspect and can be challenging to the player and reveal learning factors that help said player solve the maze in an efficient way. In general, mazes have different factors that can change the player's process in finishing the maze. Factors such as the type of maze,  and incentives should be considered when solving a maze. In our project, we wanted to understand what  type of algorithm would be  efficient when considering these factors. Throughout the year, we researched the types of algorithms used when solving a maze and incorporated these algorithms into our code.  Specifically, we analyzed  the mouse junction algorithm, wall following algorithm, Dijkstra's algorithm, and the Tremaux algorithm. With our research, we predicted that Dijkstra's algorithm would solve the maze in the quickest and most logical manner. This is because this algorithm determines the quickest route, based on the value of each maze square. It is a more refined algorithm in comparison to the other algorithms as it purposefully selects a route. We then implemented algorithms into our code and created a player that would navigate through a simple maze with each of the algorithms. We measured efficiency based on the time and number of tiles that the player takes to complete the maze. Our end goal is to give the information and research we learn to  scientists in the psychology field as a basis for them to expand their experiments on. We expect our results will allow them to choose a more efficient algorithm for their experimental needs.

# Problem:

Within the psychology field, mazes have been used heavily for experiments regarding decision-making and cognitive processing. Although there is a clear connection between mazes and the psychology field, we found that there is a minimal collaboration between programming and psychological studies, leading to a limited extent of knowledge [6]. Nicole D. Anderson speaks to this issue from a pedagogical perspective stating “computational thinking has historically been a skill that is exclusively taught within computer science… Psychology is an excellent example of a discipline that would benefit from computational thinking skills because of the nature of questions that are typically asked within the discipline” [12]. This demonstrates a disconnectivity between the sciences. It’s imperative that as we advance technologically, we learn how to transfer these new knowledges to other areas of knowledge, expanding upon what was previously possible. Psychology experiments have a history of using mazes to gauge human and animal decision making. A neobehaviorist Edward Chace Tolman claimed “Everything important in psychology…can be investigated in essence through the continued experimental and theoretical analysis of the determinants of rat behavior at a choice-point in a maze." [13]. Mazes are an important aspect of psychological investigations as they can be a great visualization for cognitive thinking. Algorithms can be built to solve these same mazes for more specific purposes. Comparing the results of this data could be beneficial to better understand the purpose behind human choices.

# Objectives:

Through our code, we are investigating the union of technology and social sciences. We are determining if (as the Anthropocene becomes more digitally-minded) we can harness the effective visualization and data collection methods of coding to bolster psychological research. By the visual means of mazes, we will be demonstrating the overlaps of algorithmic logic and the inconsistency of human-made decisions.

To start our project, we wanted to investigate mazes and the types of algorithms that are utilized to complete them. For our program, we decided on the random mouse algorithm, the wall following algorithm, Treumeux's algorithm, and Dijkstra's algorithm. We chose these algorithms to display a variety of solutions to solve a maze. Each of these mazes has a unique component to them that can be an important factor in how they solved the maze. We also wanted to implement these algorithms with the idea that the program will be going through the maze with no prior knowledge of what paths to take to complete the maze. This helped us simulate the human aspect of our code to demonstrate how humans would utilize these algorithms to solve this maze. We also chose these mazes because our program can display how each of these algorithms works and its efficiency based on the number of tiles and time it takes to complete the maze.  We want to use our results to solve real-world problems that can connect the use of mazes and psychological sciences to best represent how decisions can impact efficiency in completing tasks.

# Research:

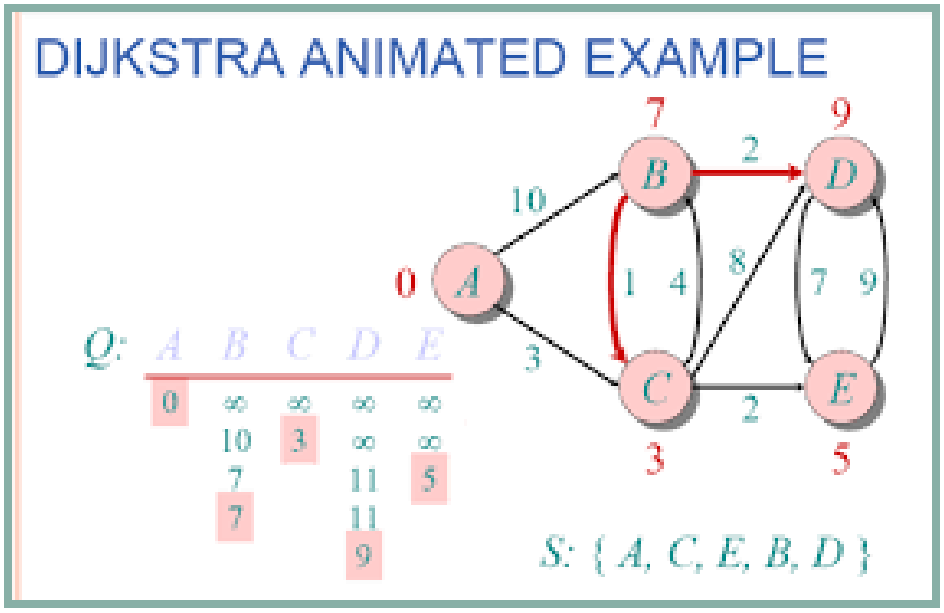
Human decision-making is a big factor when it comes to psychology and everyday thinking. When faced with a decision, there are a series of strategies and methods that are taken into account in order to arrive at a specific decision. In human decision making these methods include a single feature method, the additive feature method, and the elimination by aspects method. In the single feature method, one central variable is taken into account and is effective in simple decision-making. In the additive feature method, it takes into account all aspects of a decision and choices that are made to come to the conclusion of a decision, this method is best when determining the best option for a multitude of choices. Lastly, there is the elimination by aspects method. This method is described as evaluating one characteristic of a decision at the beginning with whatever feature you believe is the most important. When an item fails to meet the criteria that are established, it is eliminated from the list of options. This list of possible choices gets smaller and smaller as you cross items off the list until you eventually arrive at just one choice [7]. These methods align closely with how decision-making within a program is taken into account and can be modeled in both humans and the program. However, decision-making within humans has other methods that rely on past experiences and images that are perceived in one's head. These methods are called the availability heuristic and the representativeness heuristic. These decision-making strategies rely on human mental instincts. For this year, we specifically took note that our program will use methods that require less mental and emotional instincts in order for the program to solve the maze in an effective and useful manner. [7]

In order to find how mazes work and what algorithms work best with which corresponding maze, we looked into types of mazes throughout history and scientific discovery. One type of maze that we looked into was an animal maze which has been used heavily to better understand cognitive processing and decision making. William S. Small used a maze to measure the learning capacity of rats to test their intelligence and long-term memory. He utilized rewards to test the rats' spatial map that they create when going through the maze which is very similar to what humans do and is something we have thought about incorporating into our code [1]. Another type of maze is the Porteus maze which was invented by Stanley Porteus and was used to measure the ability to plan ahead, deal with novel problems and improve performance with practice. This is the basic type of pen and paper maze we see with the participant looking

over the maze rather than actually being in the maze. It has been used heavily to similarly show cognitive processing in human participants and performance going through the maze [5]. The last type of maze we looked into was virtual mazes since this fits in with our project. One computerized maze was created by Elizabeth Hampson which was used to simulate the experience of being in real life. The virtual maze was used to create greater ecological and comparative validity rather than pen and paper mazes. This type of maze was then used for multiple different experiences to measure human performance and decision-making [5].

Maze-routing algorithms are methods to find a way between any two locations in a maze. People use these algorithms to achieve the fastest route with the least amount of decisions possible. Algorithms can be used to more efficiently get through a maze taking into consideration different types of mazes and paths. They can also be used for decision-making in real life. In an emergency, algorithms may help people consider quick navigation tactics [3]. For example, a nurse looking for medicine in another room in a hospital has to essentially go through a maze in order to get from her location to her destination. By using any maze routing algorithm, she can assess which way will be the most efficient without going into every room to check if it is the right one. She will need to chart a path, make turns when she reaches dead ends, and decide which way to go at every intersection. An algorithm can be used to make these decisions and possibly more effectively get through this real-life “maze.”

Dijkstra's algorithm is more advanced as it places a value on each unit of space to locate the quickest path from the start to a target. It was invented by the Dutch computer scientist Edsger Dijkstra in 1959. A more specific definition for Dijkstra's purpose is “finding the shortest path from a starting node to a target node in a weighted graph is Dijkstra’s algorithm” [14]. This algorithm replicates more GPS navigation as routes to target destinations are calculated based on finding the fastest way. Each road is given the value of miles and speed limit and this is all factored into the GPS or human calculated decision for which way to drive.




In **Figure 1** we can see how decisions are determined based on values. Dijkstra's algorithm typically finds the quickest path possible. So in this example, to get from A to E, Dijkstra will choose to go from A to C to E as it is the lowest valued path [14].

Tremaux's algorithm was created by Charles Pierre Tremauz in order to solve complex types of mazes. Within this algorithm, the player will walk through the maze and take any path at an intersection it has never seen before. It then continues forward until it reaches a dead end and then will turn around and walk back the same way it came. Once it reaches an intersection it has been to before, it will take a new path instead of the one it already went on. This means it will have to keep track of which path it has already been down at each dead end [8]. This algorithm is an efficient method to get through a maze and will always get through a maze. However, it will not always give the player the shortest path through a maze because it has to continually go back when it reaches a dead end. This algorithm will lead to the player going back through paths that lead to dead ends multiple times which will, in turn, create more time to go through the maze. This algorithm is pretty effective because it will have a 50/50 chance of selecting the right path when at an intersection it has never seen before, however it does require that the player record a mark at every intersection.

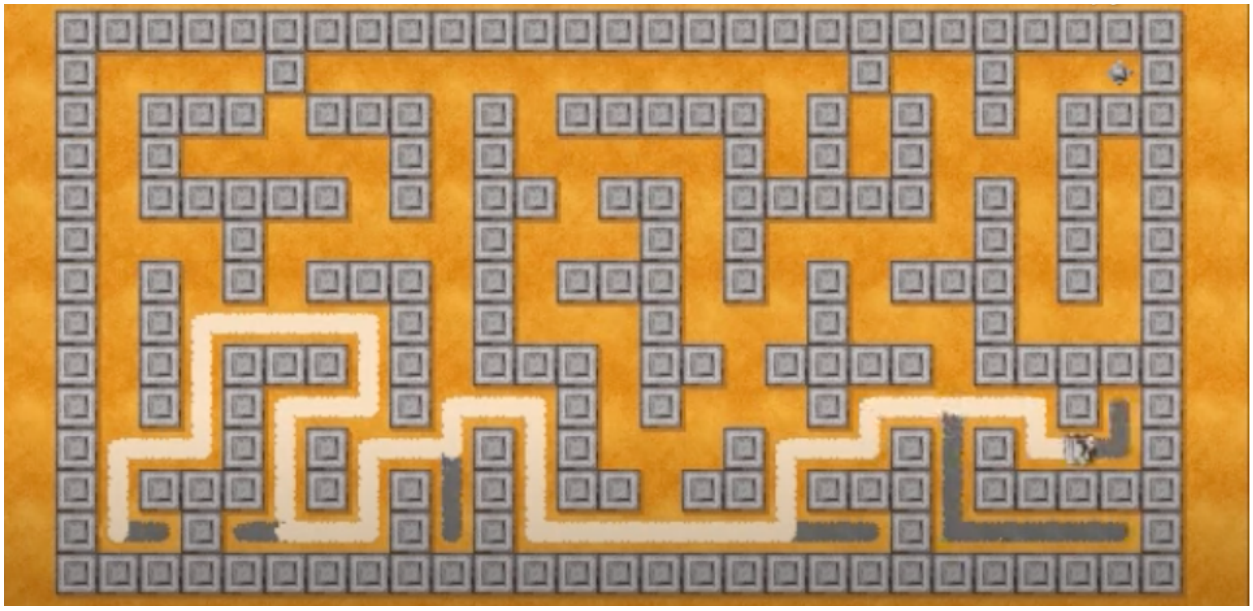
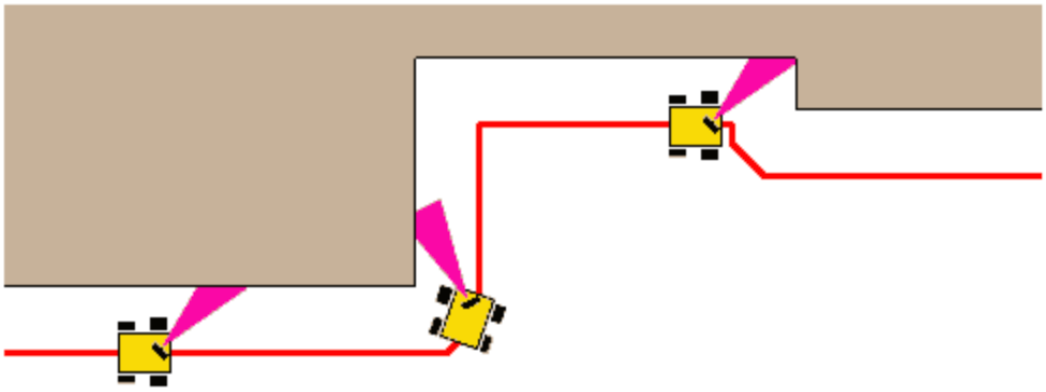


**Figure 2** shows a path drawn while following the Tremaux algorithm to go through a maze. The gray lines are paths that led to dead ends and the yellow lines are paths that eventually went over the gray lines and led to the end of the maze [8].

The Mouse junction algorithm was created based on the fact that an unintelligent mouse can complete a maze with this method. This method consists of proceeding in a straight line until a junction is reached and then making a random decision as to what path to follow next. Although theoretically such a method would always eventually find the right solution, it is extremely slow. This is because of its repetitive nature which is not always the most efficient. This method can demonstrate a human's ability to go through a maze without any prior knowledge of the maze and requires little to no memory. [11]

In the wall following algorithm, the player will essentially just follow along the wall on either its right or its in left as it goes through the maze. If the player follows along the wall on its left it will use the left-hand rule and if it follows along the wall to its right, it will use the right-hand rule. It would be as if someone were to go through a corn maze holding their arm out to the side of them turning at each intersection where their arm will continually be touching a wall. If using the right-hand rule, the player will turn at every intersection that is on the right side of the path. Even though this algorithm may not be the most effective because the player could end up taking all the wrong paths, it will always lead to the end of the maze so long as it keeps moving forward. When it reaches a dead end it will still follow the wall and get out of the dead-end which will ensure it never just gets stuck in a dead end and stops moving [10]



In **Figure 3** a truck's light is shown following the wall closely as it turns and proceeds with its route. The path of the truck is parallel to the wall as shown with the red line [9].

# Methods:

**Start:**

Through the course of this year, our project goals evolved. At the beginning of the year, our project started with an interest in the psychology field. We originally wanted to analyze how humans make decisions in quick-paced situations that are presented to them on the spot. We also wanted to look into the different personality types and what sort of decisions this influences. Through doing research, we discovered that mazes have been used heavily for experiments regarding decision-making and cognitive processing in the psychology field. This led to our decision to use a maze as the basis of our project as it involves a surplus of different decisions at each intersection. However, this plan called for human participants and we were unable to contact the institutional review board with enough time to use participants within this year. We then decided that in order to better understand decision-making as a whole we could still use mazes and test the efficiency of different algorithms.

**After Narrowing Down Project:**

After making the decision to test the efficiency of algorithms going through mazes, we then moved forward with our project. Our next steps were deciding what we needed to research as well as how the code would work to represent this question. We researched the use of different types of mazes in the psychology field and how they have been used in various experiments through the years. We also thought about how we could equate these different types of mazes to our project using python. This led to our decision to use a Pygame import in order to show a maze using a matrix. We also did research on different types of algorithms used to go through a maze and found the mouse junction algorithm, wall following algorithm, Dijkstra's algorithm, and the Tremaux algorithm. We decided we would use these to go through a set of different mazes to find which one is the most efficient based on the time it takes or the number of moves to get from one side of the maze to the other. We first started designing what our mazes would look like on paper and then figured out how to translate this into the code. We think that the results of these mazes could be shared with psychology researchers for them to then use in experiments with possible human participants.

# Python and Pycharm:

In our program, we wanted to model a basic maze structure and apply the four algorithms to see how efficient each algorithm is in getting the player from the start of the maze to the end. We are testing efficiency by using time and the number of tiles that the player takes to get to the end of the maze. For the basic program, we created a grid structure as the maze for all four algorithms. To start, we implemented a pygame in order for us to build and use a maze. This pygame was used to help us visually create our program and its components.

For this year, we primarily focused on two algorithms, the Mouse Junction Algorithm, and the Wall-following algorithm. Ultimately, we used these algorithms to start our program and model the premise of a maze. After further research, Dijkstra's Algorithm was not used in our program because of its algorithmic nature, where the program has knowledge of the entire maze before it solves it. This was not relevant to our problem as we were using the program to solve it similarly like a human. Finally, due to time constraints we did not fully develop Tremaux's algorithm in order for the player to solve the maze using this method.

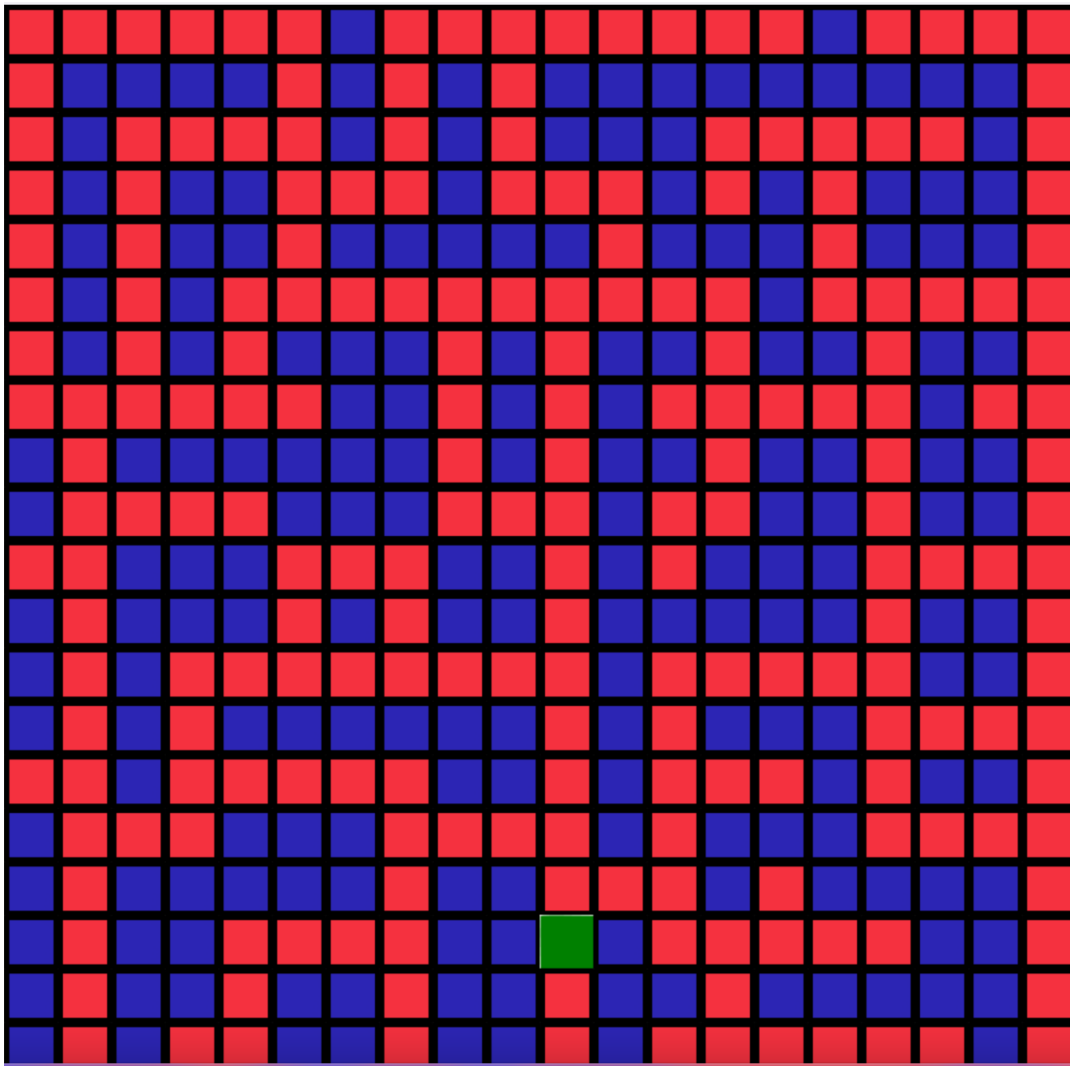
```
class maze:

    def __init__(self):
        self.initialized = False
        self.running = False
        self.window = None
        self.background_png = None
        self.tile_size = (40, 40)
        self.num_x_tiles = 20
        self.num_y_tiles = 20
        self.windowWidth = self.tile_size[0] * self.num_x_tiles
        self.windowHeight = self.tile_size[1] * self.num_y_tiles
        self.player_column = 0
        self.player_row = 0
        self.number_moves = 10
        self.player_direction = 'right'
        self.directions = ['up', 'right', 'down','left']
        self.following_direction = {'right': ['down', 'up'],
                                    'left': ['up', 'down'],
                                    'up': ['right', 'left'],
                                    'down': ['left', 'right']}
```

In **Figure 4,** it shows how we defined our variables in a class. This was the general base of our code and created the premise for it.

```
self.maze = [[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
             [0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
             [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0],
             [0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0],
             [0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0],
             [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
             [0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0],
             [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
             [1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0],
             [1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0],
             [0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0],
             [1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0],
             [1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0],
             [1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0],
             [0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0],
             [1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0],
             [1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0],
             [1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0],
             [1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0],
             [1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
             [1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0]]
```

In **Figure 5,** it shows an array of the rows and columns within our maze. The 1's describe the walls of the maze, and the 0's describe the pathways.



In **Figure 6,** this is our final maze that we used to apply all of our algorithms. This visually validates our program by showing the player and its method on navigating throughout the maze, based on the algorithm chosen.

**Random Algorithm**

In the random algorithm, we applied a random package to get the player to move randomly. As the code runs, the player will navigate through the maze. If there is a dead-end or the player hits a wall, the player turns around or makes a random decision as to what path it goes into next. This method is simple, however, has been found to take a longer amount of time because the player can repeat the same paths that it has already passed.

```
    def random_algorithm(self, directions_moves, number_paths):
        for direction in self.directions:
            # print(direction)
            if self.player_direction == direction:
                #   print(self.player_direction)
                # checking if there is a wall in direction of the player's direction
                if directions_moves[self.player_direction]:
                    while True:
                        self.player_direction = self.set_direction()
                        if not directions_moves[self.player_direction]:
                            break
                elif number_paths >= 3:
                    while True:
                        self.player_direction = self.set_direction()
                        if not directions_moves[self.player_direction]:
                            break
```

In **Figure 7,** It shows the random algorithm function.

**Wall Following Algorithm**

In the wall following algorithm, the player is going the current direction and “holding” their arm out to the right. Whenever a junction is reached, the player will go in its right hand direction. For example, if they are going right, the wall direction will then be changed to the downward direction. This method is effective in ensuring that the player is not repeating the same paths already taken.

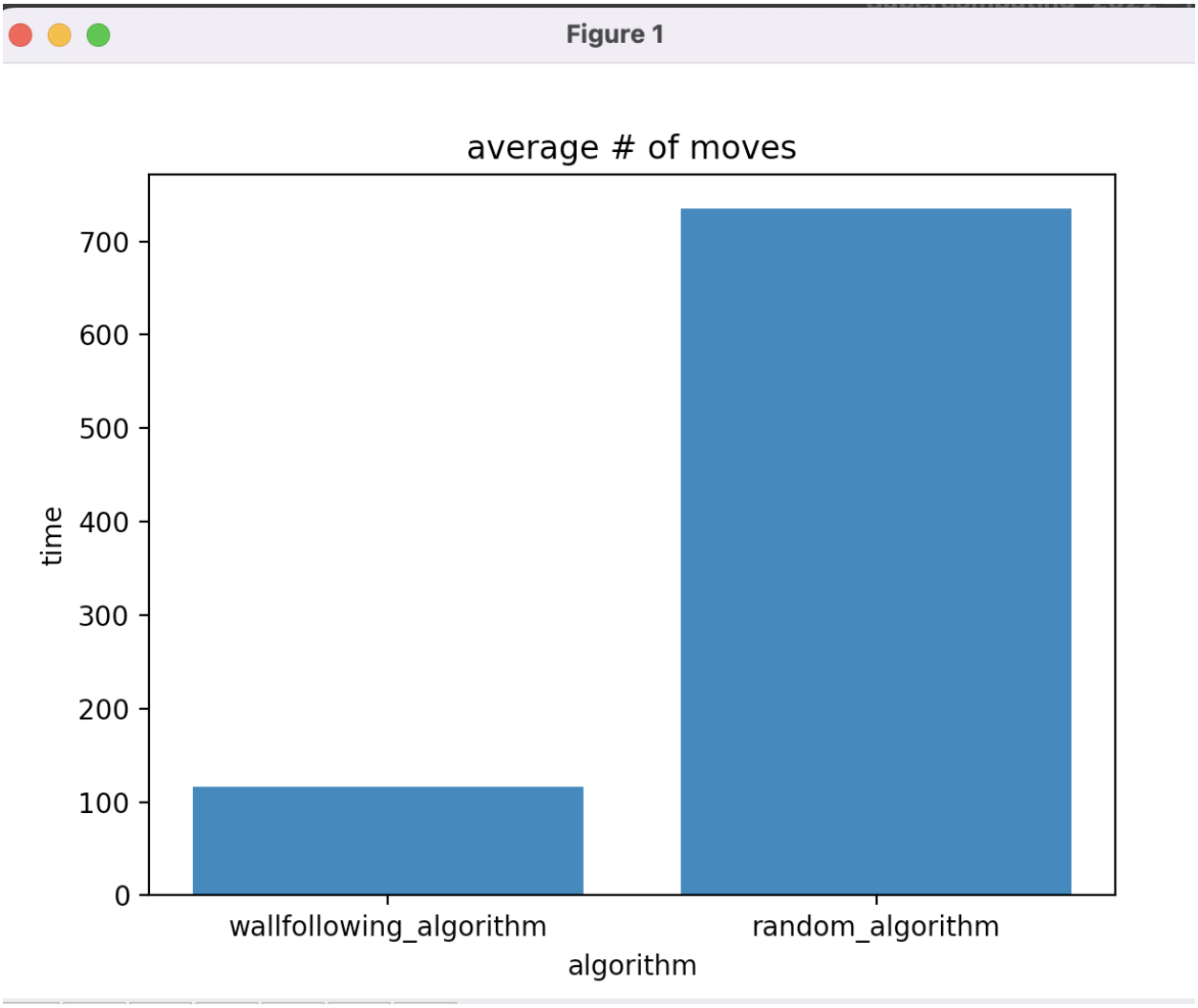
```
def wallfollowing_algorithm(self, directions_moves, dead_end):
    # if the player hits a wall in their current direction or if we hit a path when there should be a wall
    if directions_moves[self.player_direction] or not directions_moves[self.direction_wall(self.player_direction)]:
        if dead_end:
            self.player_direction = self.check_direction()
        else:
            for direction in self.following_direction[self.player_direction]:
                if not directions_moves[direction]:
                    self.player_direction = direction
                    break
```

In **Figure 8**, It shows the wall following algorithm function

```
algorithm_choice = 'wall_following'
if algorithm_choice == 'wall_following':
    self.wallfollowing_algorithm(directions_moves, dead_end)
elif algorithm_choice == 'mouse_junction':
    self.random_algorithm(directions_moves,number_paths)
```

In **Figure 9,** it shows how we created a function to allow us to run the different algorithms within our program. This helped us organize and distinguish which algorithm was running at the time.

# Results: Tables and Graphs




In **Figure 10**, it shows a bar graph of the time it took for each algorithm to complete the maze. Since the random algorithm is randomly generated every time, we did three trials of the algorithm and found its average in order to compare it to the wall following algorithm.

```
randomnumber_moves = [810, 1104, 290]
random_times = [1620, 2208, 580]
average_moves = sum(randomnumber_moves)/ len (randomnumber_moves)
print(average_moves)
wallfollowing_totalmoves = [116, average_moves]
print(wallfollowing_totalmoves)
total_times = [232]
algorithm_names = ["wallfollowing_algorithm", "random_algorithm" ]
figure, axis = plt.subplots()
axis.bar(algorithm_names, wallfollowing_totalmoves)
plt.title('average # of moves')
plt.xlabel('algorithm')
plt.ylabel('time')
plt.show()
```

In **Figure 11,** it shows how we implemented our bar graph to display our results, as well as how we found the average number of moves of the random mouse junction algorithm

# Results:

In our results, we found that the wall-following algorithm was more efficient in completing the maze in a reasonable amount of time and number of tiles taken. Based on our bar graph, we found the average time it took for the random mouse junction algorithm was 1469.3 while the wall following time, on average, was 232. The average number of moves for the random mouse junction algorithm was 734.6 moves, and the wall following method was 116. Though we were not able to test the other two algorithms that we researched, we found that this algorithm can work effectively both on the program and in real life. These two algorithms allowed the program to not have any prior knowledge of the maze beforehand, making it relatable to a human solving this maze. We also want to take into account the maze size and type of maze. In our program we used a 20 by 20 grid, and a labyrinth type of maze. This could have impacted the results of our maze, because the structure of the maze was important for the wall following algorithm, as it cannot work if there were more pathways, because of its reliance on the right-hand wall. Overall, with the type and size of the  maze taken into account, the wall-following algorithms worked the most efficiently in our program.

# Conclusion:

This year we were only able to incorporate the wall following algorithm and random algorithm into our code. While looking at both of these algorithms we found that wall following was the most efficient out of the two because it took the least amount of moves to get through the maze on average. The random algorithm ended up taking a lot more moves because it repeated paths that it had already been on and had an unlimited amount of paths that it could choose to explore. The reason wall following might be more efficient is because it only chooses paths on the right side in our case which includes less turns depending on the type of maze. We used a labyrinth type of maze, which could have a different effect on the effectiveness of the algorithm. This program can be used for a multitude of problems that can apply to real life. For example, this program can navigate and find the quickest and most efficient way through traffic for emergency vehicles. By finding the most efficient algorithm, we can use it as a system that finds accurate methods for this form of travel. Another use for this program can help with research among participants in the psychology field. We plan to give the findings of our algorithms and mazes to researchers so that they can utilize it in their experiments. This program can be utilized with human participants to find the best route through a maze and study how humans navigate through a maze compared to a program. This can be used to study the human brain and decision making as participants navigate through the maze.

# Significant Achievements:

The most significant achievement for our team this year was communication and collaboration skills. Since we are a team of three with one of us at Del Norte High School and the other two at Cottonwood Classical, it became very important for us to keep in touch with each other and meet when possible. In order to work on our code, we had meetings twice a week online with our mentor, Flora Coleman. In addition, our team met one to two times a week in person to do research or work on code. Although we all have busy schedules with work and school, our team did a good job making sure to prioritize supercomputing when necessary and meet on our own time since we were unable to meet with the rest of the Jackson Middle School teams. We effectively worked together on our project through meeting often, using GitHub, and constantly keeping in touch with our sponsoring teachers, our mentor, and each other.

**Ayvree Urrea:**

The most significant achievement that I felt that I had this year was furthering my python skill set as well as effectively collaborating across platforms. Since this was Kiara and I's first year working with a third teammate in a couple of years it became very important that we master our team working skills and collaborate even when we were not together. In order to do this, we all created a shared github repository and met twice a week online with our mentor, and once a week in-person to work on our project. I felt that I got better at pushing what I worked on to the repository so that Kiara and Violet could see what I added and also update my project whenever others made changes. Along with this, I think that my python skills grew this year as I became more familiar with the way pycharm works and the language itself. I think that I learned a lot about matrices, functions, loops, and if/else statements. In addition to these accomplishments this year, I also applied for an Aspirations in Technology Award at NCWIT and got the Honorable Mention Award.

**Kiara Onomoto:**

This year, my most significant achievement would be furthering my experience in Python and being able to communicate this skill with my mentors and teammates. Since this is my fourth year learning and implementing my project in Python, I have been able to understand and grasp the syntax and concepts that the language has to offer. With the help of my mentors, I have been able to learn how to code using Python faster and more effectively than ever. This growth has helped my confidence in presenting ideas and problems to my mentors and

teammates in order to find solutions that are effective for our projects. In addition to this, I am proud of my Aspirations in Technology award by NCWIT, where I achieved the rising star award. This has made me appreciate the opportunities that coding and Supercomputing have given me.

**Violet Kelly:**

I've had a plentiful sum of significant achievements (and lessons) this year as it is my first in the Supercomputing program. I believe my most significant would be my developed understanding of pycharm and its unique language as well as my general understanding of code and how it can be relative to really anything. This program is a great way to immersively grasp how computational thinking can solve real world issues. Through the research and code necessary for our project, I've come to learn a lot about the facets of code and how they can fit into the more traditional sciences. For example, code can be incorporated into psychological experiments to represent certain scenarios (in our instance a maze regulating human decisions). As for my achievements in python language, with the vital aid of our mentor, I've come to recognize the applications for if/else statements and defining functions and their respective scopes. To conclude, I'm very glad to have joined this program, although late in the game, as it's not only taught me the specifics and significances of computer science but also an enriched perspective on how to tackle complex world issues.

# Acknowledgments:

Karen Glennon, Retired Teacher

Patty Meyer, Retired teacher

Flora Coleman, Sandia National Laboratories

# Bibliography:

[1] Goodwin, Dr. C. James. “A-Mazing Research.” *Monitor on Psychology*, American Psychological Association, Feb. 2012, www.apa.org/monitor/2012/02/research.

[2] Mohseni, Fahimeh, et al. “A Review of the Historical Evolutionary Process of Dry and Water Maze Tests in Rodents.” *Basic and Clinical Neuroscience*, Iranian Neuroscience Society, 2020, www.ncbi.nlm.nih.gov/pmc/articles/PMC7878036/.

[3] Pullen, Walter D. “Think Labyrinth: Maze Algorithms .” *Think Labyrinth: Maze Algorithms*, http://www.astrolog.org/labyrnth/algrithm.htm#solve.

[4] “Maze Solution #3 - Tremaux's Algorithm.” *Maze Solution #3 - Tremaux's Algorithm: V19FA Intro to Computer Science (CIS-1100-VU01)*, https://vsc.instructure.com/courses/6476/pages/maze-solution-number-3-tremauxs-algorithm?module_item_id=713459.

[5] “Historical Mazes.” *Maze Engineers*, 6 Dec. 2020, https://conductscience.com/maze/historical-mazes/.

[6] American Psychological Association. (n.d.). *A brief introduction to python for psychological science research*. American Psychological Association. Retrieved February 26, 2022, from https://www.apa.org/science/about/psa/2019/07/python-research

[7] Cherry, Kendra. “How Time, Complexity, and Ambiguity Influence Our Decisions.” *Verywell Mind*, Verywell Mind, 10 Nov. 2019, www.verywellmind.com/decision-making-strategies-2795483#:~:text=When%20making%20a%2

0decision%20in,make%20decisions%20and%20judgments%20quickly.

[8] “Tremaux Algorithm.” *Maze Solution #3 - Tremaux's Algorithm: V19FA Intro to Computer Science (CIS-1100-VU01)*, vsc.instructure.com/courses/6476/pages/maze-solution-number-3-tremauxs-algorithm?module_item_id=713459.

[9] “Wall Following.” *robocon16:Programming:wall_following [RoboWiki]*, students.iitk.ac.in/robocon/docs/doku.php?id=robocon16%3Aprogramming%3Awall_following.

[10] “Maze Solving.” *ArcBotics*, arcbotics.com/lessons/maze-solving-home-lessons/.

[11] “Maze Classification.” *Think Labyrinth: Maze Algorithms*, http://www.astrolog.org/labyrnth/algrithm.htm.

[12] Anderson, Nicole D. “A Call for Computational Thinking in Undergraduate Psychology.” *Psychology Learning and Teaching* , 2016, journals.sagepub.com/doi/pdf/10.1177/15248380211069059.

[13] Goodwin, C. J. (2012, February). A-mazing research. *Monitor on Psychology*, *43*(2). https://www.apa.org/monitor/2012/02/research

[14] “How Does Dijkstra's Algorithm Find Shortest Path?” *Stack Overflow*, 1 Feb. 1968, stackoverflow.com/questions/61258237/how-does-dijkstras-algorithm-find-shortest-path.