A-Maze-Ing

New Mexico

Supercomputing Challenge

Final Report

JMS32

Jackson Middle School

Team Members

Brendan Kuncel

Quentin Dye

Teacher

Karen Glennon

Project Menor

Patty Meyer

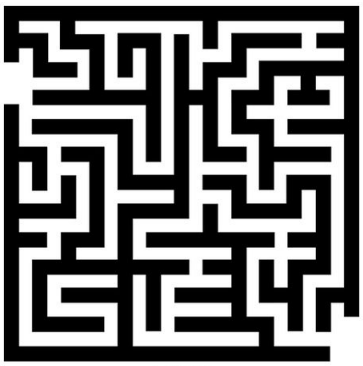
Executive Summary

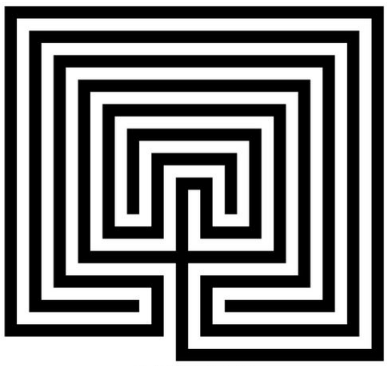
The first ever recorded labyrinth was found in Egypt in the 5th century, and it was discovered by Herodotus.[1] Unfortunately this is not a true maze, it was a labyrinth, as hedge mazes only started to occur in Belgium 13 century. The word "Maze" was also first seen in the 13th century, and it meant the feeling of delusion or bewilderment, and that connects to the idea of "Amazement" or to "Amaze."[1] So Mazes have bewildered many people tracing back all the way to the 5th century, and surely there are quite a few ways to solve one. So what is the best way to solve one, there are thousands and thousands of algorithms that can solve a maze, but which one can solve them the fastest and with the most success. The question could be applied in a couple of different situations like navigating city streets. Before we got into that question we needed to do a little bit of research, so we first explored the types of mazes that we could potentially solve. After doing that research we found that there are quite a few type of mazes that we could use in our project. One of which being regular two dimensional mazes, that could be drawn out onto a piece of paper, and these types of mazes can be as complex or a simple as you want. The Second type of maze we looked at was labyrinths, and they are defined as a complex network of passages that are difficult to traverse.[2] Although that sounds a lot like a maze labyrinths and mazes do have a large difference, as mazes usually include complex branching, and labyrinths being more unicural, only having a single path to follow with one entrance and one exit. The other classes of mazes get even more complicated like three dimensional mazes, as three dimensional mazes have 6 different directions of travel. The other important thing to research is how we are going to solve the mazes, which include things like the wallflower method, the flood method and Dijkstra algorithm, each one having a strength and weakness.

Wallflower being useful for regular two dimensional mazes that don't have any loops, as loops are one of the weaknesses that wallflower method have. Dijkstra method thrives in situations regarding loops, so this Algorithm would work quite well for mazes that include loops. So our project aims to explore the options you have in a situation where you have to navigate your way around a city without any map or any external help.

Statement of the Problem

Mazes are a quite complicated and interesting, as they can be as complicated as the program that runs your phone or as simple as a hello world program built in python. So how do we solve them? Well first we should look at what kinds of mazes we are dealing with or what maze you want to solve, as there are regular two dimensional mazes, labyrinths and three dimensional mazes. Then you need to know how you can solve the maze, your first thought may be the left hand method or a completely random method. The question is what is the most effective method you could have. Our project aims to answer the question of what method is the most effective or what can solve the maze the fastest. We decided that we would stick to the traditional maze and a city block type maze. We decided that we would try and choose 3 to 4 different methods to test which would be the fastest or the most effective. Those being a Flood method, Wallflower method, and the Dijkstra Method.



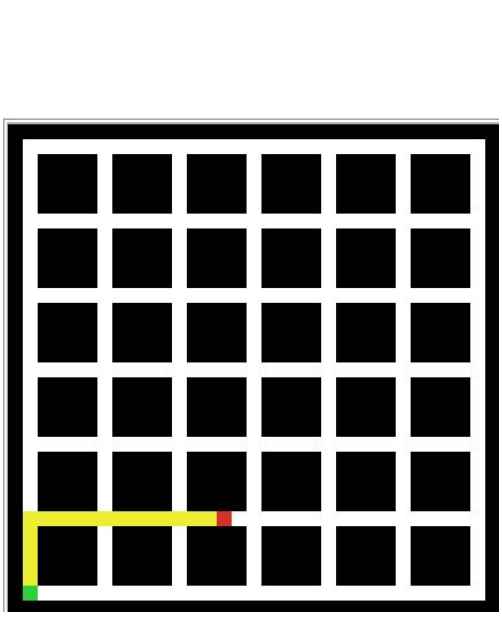


3D maze

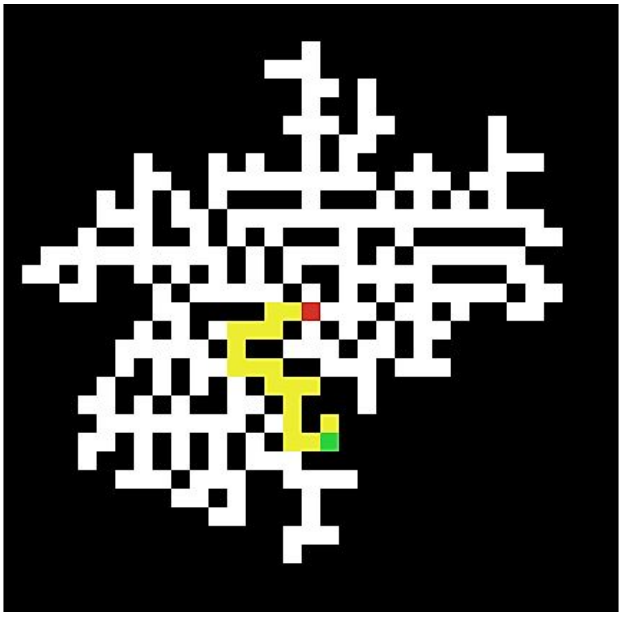
Method

The method we used was to implement a variety of maze solving and maze creating algorithms. We took some of the most common ways to solve a maze, following one wall, or just picking the path closest to the end of the maze, and put them into netlogo with two different maze types, one that was random every time, and a grid, representing a city. We used the netlogo behaviour space to run many repetitions of the different types of solutions with the different types of mazes. We then took the information and sorted it by the fastest average solver (fastest meaning in the lowest number of steps taken to the end of the maze). The steps taken are monitored by the patches themselves, as they have a variable that is assigned whenever an agent steps on them. This helped demonstrate the steps taken that are more than what is necessary for the design, which could be improved in the future by having a way to follow the right or left hand rule while ruling out the places already visited and straight dead ends (the places that you can see that it is a dead end at the start of the passage). The code used for the right hand method just checks to see if there is path next to it from right to left, and the opposite for the left hand

method. The facing strategy was based on facing the exit and turning either clockwise or counter clockwise until a turn is passed. That path is then taken until another turn is presented.



"Grid" maze



"Random" maze

Verified and Validated

Our model is representative of real world situations. It can be used in helping to find the shortest route to a place in a maze or a city. This was verified by looking at the most common ways to solve real-world mazes. This inquiry found following walls, using a map, and just going in the closest direction to the maze. While most would not consider using a map "solving" a maze, it is still a method used, and one of the fastest ways. We included this method by having both a computer solve the maze and have the other methods show the optimal route that they took (which still could not be the best route, if they go all the way around and reach the end, the

“best route” could actually be one of the longest.) As the way a person will solve a maze can be anywhere from knowing for sure to randomly running around, it is good to have a method to follow to help not be lost and confused.

Downtown Denver:

Finding your way through a modern city can sometimes be difficult and finding the best way to navigate can be quite beneficial.

Results

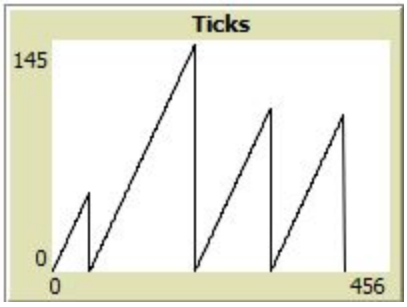
The results that we gathered, were that in a grid formation (aka what would be used in the case of a map) the best way to reach the end quickly would be to follow the method of going in the direction closest to that of the direction to the end of the maze. This did not, however hold true with a randomly generated maze, where facing the end would get there only on certain occasions. On all of the other trials, facing the end would get the agent stuck, going into a dead end, and no trying to get out, but just continually going back to the dead end. In this type of maze, the only method that would solve it 100% of the time was either the right hand or left hand method. These, however, could take very few steps, or almost every possible step, depending on if there were several branches in the best path from start to finish. The best path, however, is

shown in a fashion very similar to the “flood” solution (where the maze is solved by the computer by “flooding” the maze till an end is reached) by taking the path with the least steps and highlighting it, or giving a sort of “map” to the finish

Conclusion

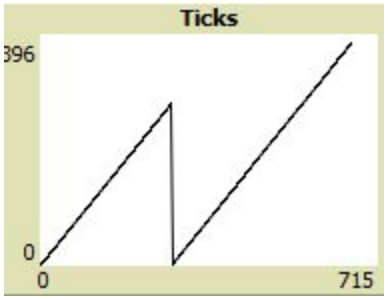
Mazes are complex problems that are just waiting to be solved, but which method is the best, well this is what our project aims to answer. The initial mazes we were looking at was random mazes, and this took us some time to get down, as the program to build a random maze is not that simple, and took some time to program. Then we have to look at what methods we are going to use to solve these mazes, and the first thought we had was the left and right hand methods or what is called the Wallflower method, but we needed to attempt to go a little deeper in terms of the solving algorithms. Then we looked into Dijkstra’s algorithm, this unfortunately did not play out, as we had trouble with moving turtles from node to node, but it works by finding the node with the shortest distance from itself, then looking for the next shortest distance keeping track of what it has already gone too and staying away from those spots. So as of now, we were only able to find that the flood method worked the best, and this makes sense as it does the entire maze and looks at every possibility and decides what was the quickest and shortest path to the finish. Unfortunately we were hoping to find a method that humans could use to find there way to the end of the maze. So we intent to continue this project and implement some more complex and creative mazes algorithms and a link based maze generator, as we want to further explore the Dijkstra Algorithm.

Data



Picture 1:

The number of steps (y-axis) for each method.From left to right, right-hand, left-hand, face the end always going clockwise when unsure, and facing the end always going counter clockwise when unsure.



Picture 2:

The number of steps in a randomly generated maze, this graph only shows the right and left hand methods, respectively. The other two methods are not shown, as they were unable to complete the maze, and made the rest of the data unviewable..

Most Significant Achievement

Brendan Kuncel- My most significant achievement on our project was the type of coding I had to try to understand and write. The whole point of the project is to have several different methods to perform the same action, so it was quite a strain to come up with the code for several ways to solve a maze.

Quentin Dye- My most significant achievement on our project was diving deeper into research, as this was quite an important part of our project. I found a couple of different methods to solving mazes which helped us to look at what method is the most efficient, and we found that

some are better than others but some of that also depends on the type of maze we have. The research done helped to drive our project in more focused direction, and I think I did a great job looking into and analysing what we found for our project.

Acknowledgements

Karen Glennon - Ms.Glennon was a big help in a couple of things throughout our project, as she always read all of our papers, and checked for spelling and grammar. She also looked at our program to make sure that it made sense to her as other people are going to need to understand it other than ourselves. Ms.Glennon was a great help in making sure that our work was presentable, and for that we thank her, and acknowledge her help in our project.

Patty Meyer - Ms.Patty helped in quite a few things as well, the first and most important would be that she would help us talk through an idea, and help us to develop our own Ideas. This was big help to us as our idea was quite unclear in the beginning as first we started with a project about entropy then moved towards something we actually liked, Ms.Patty helped us to come to the idea of mazes and what the most efficient way to solve them is. The second thing would be helping us in our research, as she suggested that we look into a couple of algorithms to test in our simulation. Ms.Patty was always there to help us bounce our ideas off of her, and helped us to develop those ideas, and for that we thank her, and acknowledge her help in our project.

References

Pullen, W. D. (n.d.). Maze Algorithms. Retrieved November 27, 2017, from http://www.astrolog.org/labyrnth/algrithm.htm

“Cris' Image Analysis Blog.” *Cris Image Analysis Blog RSS*, www.crisluengo.net/index.php/archives/277.

“Maze Generation.” *Maze Generation - Rosetta Code*, www.rosettacode.org/wiki/Maze_generation

“Programming Theory: Solve a Maze.” *Algorithm - Programming Theory: Solve a Maze - Stack Overflow*, www.stackoverflow.com/questions/3097556/programming-theory-solve-a-maze

“Maze Solving Algorithm.” *Wikipedia*, Wikimedia Foundation, 17 Feb. 2018, en.wikipedia.org/wiki/Maze_solving_algorithm.

Rathor, Shantur, et al. “Maze Solving Algorithms.” pp. 1–15.

[1] Krystek, Lee. “Amazing Mazes”. The UnMuseum. 2001

[2] Oxford Dictionary. “Labyrinth”