Developing a Micro Mouse and A Random Generating Maze Using Machine Learning and Artificial Intelligence

Written by Kerem Guventurk and Mert Karakas

Abstract

As computer science courses and the maker movement engages youngsters and students, the demand for an education concentrated on computer science and engineering is vigorously increasing. Data from the National Center for Education Statistics; analysis made by the American Statistical Association, ranging from 2010 to 2013, shows an increase of demand in STEM fields like Computer Programming (77.2%) and Computer Info Tech Administration and Management (92.0%)(Myers). The number of hack-athons, maker fairs and robotics competitions around the world is also increasing which draws more competitors, motivating more people to learn how to connect a simple circuit, how to use open-source platforms. From last year, the increase of maker fairs organized worldwide has increased more than 219,000 sites (Media Kit). Most innovative technologies at home and industry are electronic and mechatronic devices which have integrated algorithms to make daily life easier. Research and developments in concepts like Artificial Intelligence and Machine Learning are what makes these devices so helpful and efficient.

1 Introduction

Artificial intelligence and machine learning have been two of the more popular, if not the most two popular, subjects in Computer Science today. The products of these two phenomena are everywhere in our daily lives, from minor pop-up adds that seem tailor-made for you to wide-range systems such as Siri, from enjoyable computer chess games to specific GPS monitors, from pedestrian vending machines to the infamous Mars Rovers. They are so immersed in our daily lives that, we take these machines for granted, wondering how people lived without these smart gadgets. In order to honor these subjects and to learn more about them, we, the Hisar Computer Science Club, initiated a project to construct a micro-mouse that navigates a two layered 120cmx120cm random generating maze using artificial intelligence.

Before expounding on our project, I would like to explain the meaning of our two key concepts: artificial intelligence and machine learning. “Simply put, artificial intelligence is a sub-field of computer science. Its goal is to enable the development of computers that are able to do things normally done by people -- in particular, things associated with people acting intelligently.”(Kris Hammond). “Machine learning, our second key term, is also a subfield of computer science, which has evolved from the study of pattern recognition and computational learning theory in artificial intelligence.” (C.M Bishop). Machine learning explores the study and construction of algorithms through a given set of data. So, it basically gives a prediction by processing a set of already given information. In our project, we have used these concepts to develop an “artificially intelligent” micro-mouse that finds its way through a maze by the process of trial-and-error.

2 Research

Before constructing the exterior of our micro-mouse, we needed to decide on the maze-solving algorithm, over various, qualified maze-solving methods, that we were going to implement in our robot.

2.1 Wall follower Algorithm

As the name of the algorithm implies, the robot keeps an eye of its surrounding. This means that whether there is an opening for the robot, it chooses that path. While the robot takes different paths, the robot records the walls that have been encountered and maps out the maze. This algorithm works best in a situation where the walls are connected to each other. The robot will have a tough time when it encounters a 4-way road since there would be no walls to follow in the

center of the road. This algorithm has two subsets in terms of solving the maze: “Left Hand Rule” and “Right Hand Rule.”

2.1.1 Left Hand rule – LHR

“Left Hand Rule” is defined that the robot would follow the path of the maze referencing the walls on the left side of the robot. Even if the LHR leads the robot to a dead end, the robot will travel to that point regardless. This rule does not aim perfection or does not aim to find the shortest distance. The LHR with the “Right Hand Rule” is designed to map the maze and calculate the shortest path. A helpful demonstration of the flow chart can be seen in Figure 1.

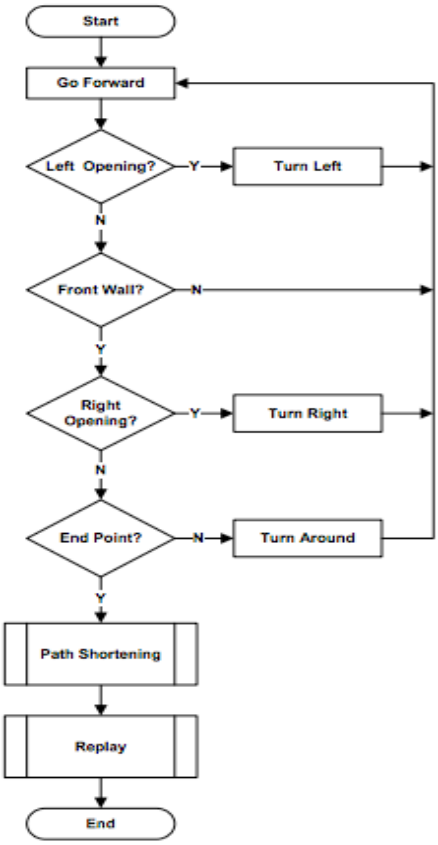


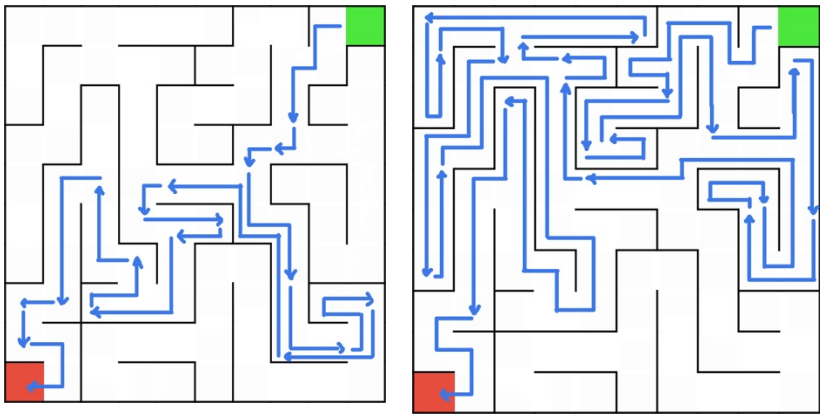
Fig 1: Flow-chart demonstrating the decision making process of the LHR. (Bakarsayutisaman)

2.1.2 Right Hand Rule – RHR

The principle of the “Right Hand Rule” is similar to LHR but instead the RHR prioritizes turning right instead of left. Comparison of the LHR and the RHR can be seen Figure 2 for a specific scenario.

Figure 2: As seen in the diagrams below, the LHR is not the most efficient method for some scenarios. Furthermore, as seen in the RHR, it is not efficient as well. (Bakarsayutisaman)

Left Hand Rule



Right Hand Rule

Figure 2: Description of Left Hand Rule and Right Hand Rule

2.2 Flood Fill – FF

Flood Fill algorithm floods the maze in order to map it. This algorithm needs a starting point and a target point in order to function. The code creates a two-dimensional array in order to record the "cells", movable spaces, From the starting point, the algorithm starts to record the cells next to the starting point. Searching for the target cell, the algorithm adds each cell to the two-dimensional array, whether the cell is a wall or an empty space, until it reaches the target cell. However, the movement of the robot is not random. The algorithm guesses the distance between the target cell and the current cell. This maze solving algorithm is the most famous and efficient algorithm yet.

2.3 Dead Reckoning – DR

Biologically backed, "Dead Reckoning" algorithm is the operation of calculating your position based on a previous location of a fixed point. Advancing in position can have a fixed speed or calculated through elapsed time. The reason why we say DR is biologically backed it that this method is used in nature by animals which is called path integration. ("Dead Reckoning")

2.4 Dead End Learning – DEL

Similar to DR, "Dead End Learning" stores dead ends that the robot enters and "virtually" places a wall in the entrance of that dead end. This applies to every dead end that the robot enters. DEL is the simplest algorithm that uses the concept of machine learning. (Willardson)

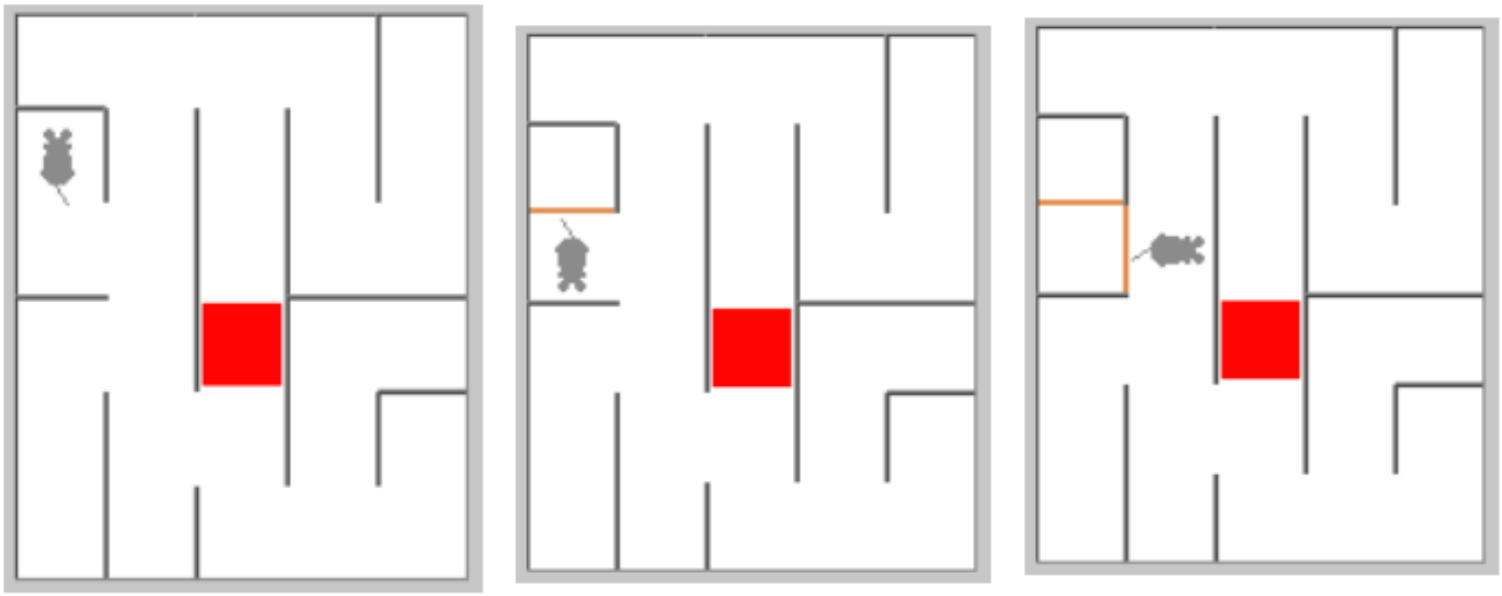


Figure 3: Description of Dead End Learning

2.5 Why these algorithms were insufficient for our project?

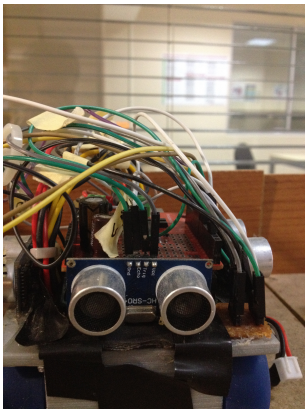
Although the algorithms that we previously shared are world known algorithms, suited to efficiently and correctly solve any maze given, they were not adequate for our project. Because our mission was to solve *unknown* mazes, some of these algorithms, which needed the map of the maze in order to solve it, were deficient. The Flood Fill Algorithm, for example, works by stacking each sell in a two-dimensional array and flooding each cell to determine the shortest route to the end point. However, in order to do that the program needs to know where the starting and ending points are and more importantly how the walls are located. Since that information is not given to

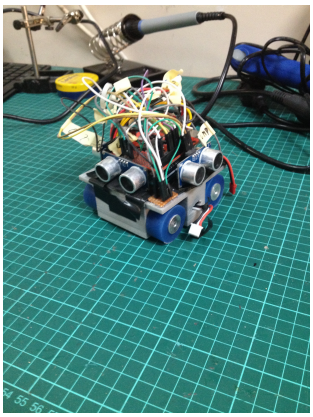
the robot and the positions of the walls are essential information to the Dead Reckoning and the Dead End Learning method, these two methods are insufficient for our use as well. That leaves us with the simplest algorithm the Left Hand Rule and the Right Hand Rule. Because these algorithms cannot find the shortest way single handedly, we decided to combine them to form a robust and efficient algorithm to solve any given randomly generated maze.

3. Development

3.1 Development of the Micromouse

Our first task was to develop a micro-mouse: a mini robot that solves a random maze through certain trial-and-error techniques. We started by designing the exterior facade of the robot in a 3-D design program named 123D Design. The school had just bought a new 3-D printing machine and we thought it would be best for us to construct the robot from our designs instead of buying some robot kit from the Internet. So, we designed our facade: a 10cmx10cmx2cm rectangle with 4 empty spots for the 4 motors of the robot. We used 6V 500Rpm micro DC motors for their affordability and efficiency, and also for the fact that two of them could be connected to each other to stimulate a single motor. After connecting the motors, we needed to link the eyes of the robot: the ultrasonic sensors to calculate how near the nearest object was. To accomplish this feat, we elected to use HC-SR04 ultrasonic sensors because they were efficient and accurate about the data they provided. After long hours of connecting and welding wires we had finally finished the exterior part of the robot, which turned out to be the easiest task of them all.





```
void forward(int powa) {
  digitalWrite(motor1A, LOW);
  digitalWrite(motor2A, LOW);
  analogWrite(motor1E, powa);
  analogWrite(motor2E, powa);
}

void backward(int powa) {
  digitalWrite(motor1A, HIGH);
  digitalWrite(motor2A, HIGH);
  analogWrite(motor1E, powa);
  analogWrite(motor2E, powa);
}

void left(int powa) {
  digitalWrite(motor1A, HIGH);
  digitalWrite(motor2A, LOW);
  analogWrite(motor1E, powa);
  analogWrite(motor2E, powa);
}

void right(int powa) {
  digitalWrite(motor1A, LOW);
  digitalWrite(motor2A, HIGH);
  analogWrite(motor1E, powa);
  analogWrite(motor2E, powa);
}
```

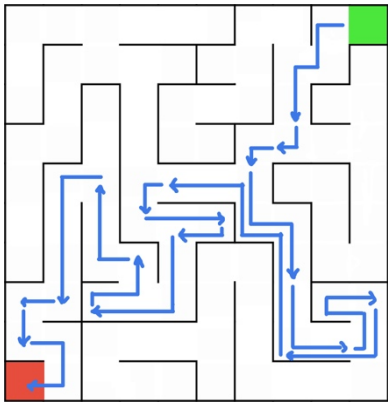
Figure 4: The micromouse's movement code

3.2 Development of the Algorithm

The interior part of the code, namely the algorithm to solve the maze, was quite difficult to both construct and integrate it into the real maze. Because the maze consisted of random challenges and orientations, the simple situational algorithms did not work for us. Instead, we needed a single robust algorithm to solve all kinds of mazes. So, we elected to use a combination of techniques. Now, the algorithm consists of 4 parts: right hand rule, left hand rule, best hand rule, and no mistakes rule. First the robot goes into the maze using the right hand rule. The primary condition of this rule is to go right whenever possible; going forward only when it can't turn right and turning left only when there are no other possibilities. The robot solves the maze eventually using this method and crosses the finish line; it counts the number of "steps" (turns and decisions) along the way as it solves the maze. Then, we manually bring the robot back to the starting point, waiting for 10 seconds for it to start the second run. This time the robot uses the left hand rule, the opposite of the right hand rule: the robot looks primarily to turn left, goes forward when it can't and

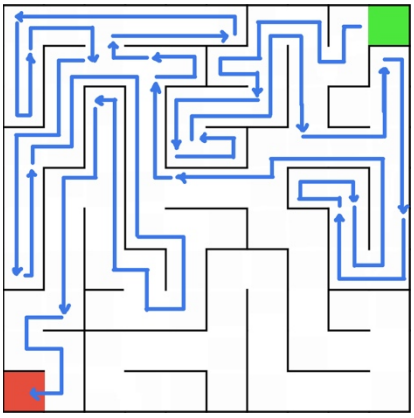
turns right only as a last resort. The robot solves the maze again using this algorithm and counts the steps. Then for the third part, the robot has to choose the shorter hand rule. Although choosing which hand rule to follow seems inconsequential at first, one needs to be careful to avoid the pitfalls present in the maze. There are many “pitfalls” in the maze, in which the correct choice in a checkpoint (points where there are more than 1 possibility to follow) leads only to a dead end, confusing the robot whether it made the right decision or not. To prevent this, our algorithm aims to minimize the number of pitfalls in the solution by choosing the shortest path possible. After the shorter rule is selected, the robot solves the maze again this time listing each move it made during the solution. It stacks the moves in a StackArray (a list library in Arduino), adding every move to the list and subtracting each wrong move. Then, after the robot finishes the third solution, we are left with the shortest solution possible to solve the maze. The robot, then, uses the stack to solve the maze in the most efficient and potent way possible.

1



Step 1: Left Hand Rule Algorithm

2



Step 2: Right Hand Rule Algorithm

3

```
on = duration4 / 10;
if (sag < THRESHOLD) {
  sagadonus = true;
}
if (sol < THRESHOLD) {
  soladonus = true;
}
if (sagadonus && soladonus) {
  baban = false;
}
//sag el kurali
if (righthandrule) {
  if (sol > 500 && on > 500 && sag > 500) {
    right(90);
    delay(240);
    right(0);
    delay(500);
    right(90);
    delay(240);
    right(0);
    righthandrule = false;
    lefthandrule = true;
    delay(10000);

  }
  else if (sag > THRESHOLD + 40 && sol > THRESHOLD + 40 && sagadonus && on > THRESHOLD - 15) {
    for (int i = 0; i < 2; i++) {
      forward(20);
      delay(170);
      forward(0);
    }
    right(90);
    delay(240);
    right(0);
    sagadonus = false;
    soladonus = false;
    righthandstack.push(4);
  }
```

Step 3: The right hand - left hand hybrid wall follower algorithm

Figure 5: Description of our hybrid algorithm

3.3 Development of the Maze

The robot, however, constitutes half of our project. We decided that a standard, static maze was not enough and thus designed a more challenging one. Consequently, we decided to construct a semi-random maze: a maze that had both static and randomly generated walls. The maze consists of 9 separate 40cmx40cm parts that, when combined, creates a 120cmx120cm one. Each one of the parts consists of 4 distinct 20cmx20cm platforms separated by 0,5 cm for the walls to be generated. The maze is a two layered one; while the first layer houses the servos

(motors that hold the walls), the second layer is the platform in which the robot drives. After the mechanical aspects of the maze were completed, we proceeded to writing the algorithm. To our luck, there were many algorithms to choose from.

3.3.1 Recursive Backtracker

This depth-first search algorithm for random maze generation generally uses a recurring backtracking method. The raw explanation of the algorithm goes like this:

"1) Make the initial cell the current cell and mark it as visited
2) While there are unvisited cells
    a)If the current cell has any neighbours which have not been visited
        i)Choose randomly one of the unvisited neighbours
        ii)Push the current cell to the stack
        iii)Remove the wall between the current cell and the chosen cell
        iv)Make the chosen cell the current cell and mark it as visited
    b)Else if stack is not empty
        i)Pop a cell from the stack
        ii)Make it the current cell "(Johnston)

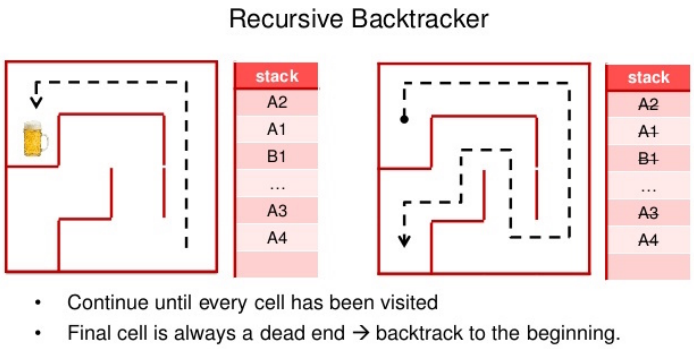


Figure 6: Explanation of Recursive Backtracker Method (Johnston)

3.3.2 Randomized Kruskal's algorithm

This algorithm is a randomized version of Kruskal's algorithm:

"1)Create a list of all walls, and create a set for each cell, each containing just that one cell.
2)For each wall, in some random order:
    a)If the cells divided by this wall belong to distinct sets:
        i)Remove the current wall.
        ii)Join the sets of the formerly divided cells." (Johnston)

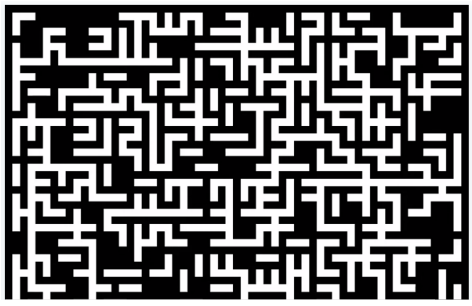


Figure 7: A maze generated by Randomized Kruskal's algorithm (Johnston)

3.3.3 Randomized Prim's algorithm

This algorithm is the randomized version of Prim's algorithm:

"1)Start with a grid full of walls.
2)Pick a cell, mark it as part of the maze. Add the walls of the cell to the wall list.
3)While there are walls in the list:
a)Pick a random wall from the list and a random direction. If the cell in that direction isn't in the maze yet:
i)Make the wall a passage and mark the cell on the opposite side as part of the maze.
ii)Add the neighboring walls of the cell to the wall list.
b)Remove the wall from the list."(Johnston)

Figure 8: The process of random maze generating by Randomized Prim's algorithm (Johnston)

3.3.4 Recursive Division method

Mazes can also be created with a recursive division method: dividing each part into two parts until the maze is completed with a clear starting and an ending point. The explanation of the algorithm is as follows:

1. "Begin with the maze's space with no walls.
2. Call this a chamber.
3. Divide the chamber with a randomly positioned wall (or multiple walls) where each wall contains a randomly positioned passage opening within it.
4. Then recursively repeat the process on the sub-chambers until all chambers are minimum sized. "(Johnston)

This method results in mazes that have long straight walls intersecting each other, making it easier to see which areas to avoid.

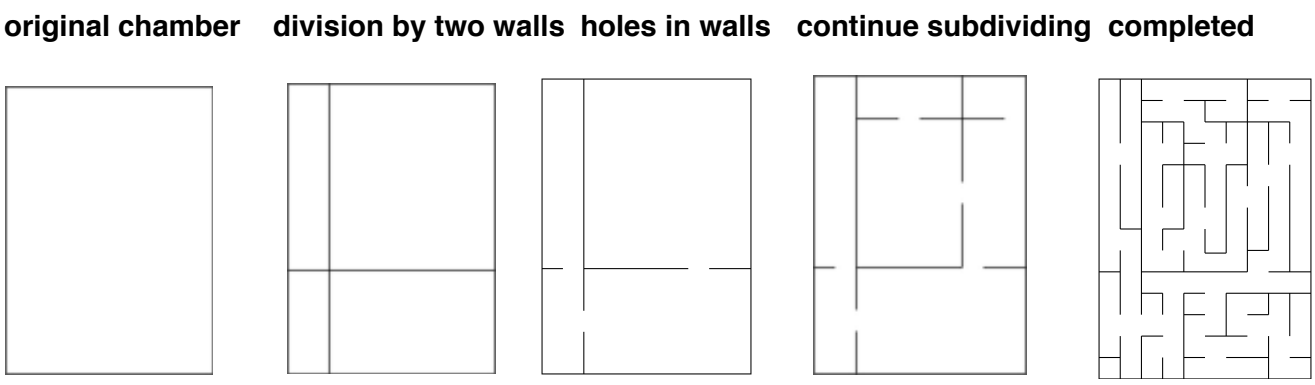


Figure 9: Illustration of Recursive Division

3.3.5 Why did we chose Recursive Backtracker Method

Admittedly, all of these random maze generation methods were viable. However, because other methods were longer than the Recursive Backtracker method and had more than one method than recursed, we chose to use the Recursive Backtracker method. We believed that it was the best algorithm in terms of efficiency, size and clarity.

```
public static void move() {
    int a = 0;
    Random rn = new Random();
    if (moves.size() > 0) {
        if (position == 1) {
            if (cube2.VisitedTrue() == false && cube11.VisitedTrue() == false) {
                a = rn.nextInt(2);
                if (a == 0) {
                    position = 2;
                    moves.add(2);
                    cube2.visited = true;
                    cube1.right = false;
                }
                else {
                    position = 5;
                    moves.add(5);
                    cube11.visited = true;
                    cube1.bottom = false;
                }
            }
            else if (cube2.VisitedTrue() == false) {
                position = 2;
                moves.add(2);
                cube2.visited = true;
                cube1.right = false;
            }
            else if (cube11.VisitedTrue() == false) {
                position = 5;
                moves.add(5);
                cube11.visited = true;
                cube1.bottom = false;
            }
```

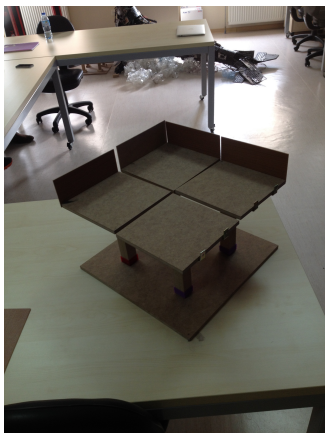
Figure 10: Our implementation of Recursive Backtracker Method

3.4 Constructing the maze

After the algorithm was finished, we were left with a 4x4 random generated maze, and using that template we controlled our servos to simulate the generated maze. If there is a wall between the gaps then the servo lifts the wall providing an obstacle to the robot, if not the wall turns backwards allowing the robot roam freely between the gaps.

Step 1: Creating each the second layer of the maze

Step 2: Combining the each 4 layers with the first layer of the maze



Step 3: Affixing the stationary walls to the maze

Step 4: Combining each of the pieces made in Step 2 to finish the maze

4. Conclusion

Upon finishing this project, we saw that this project had an exceptionally comprehensive scope in our real life. The concepts of machine learning and artificial intelligence are so prevalent that we can integrate this concept to everywhere. We thought that, for example, this robot could be well utilized in the mining industry. The mining industry can use this robot to find the path of an unknown road in the mine. By using the techniques we explained before, the robot can solve and map the convoluted road, providing the people working in the mine a comprehensive map. Also, this concept can be utilised in search and rescue mission. During catastrophic events such as an earthquake, the robot can guide his way into the debris and provide food for the needy. All and all, these concepts and the robot we designed have immense capabilities for future problem solving and we believe that they will be the most popular concepts for years to come.

5.Bibliography:

1. Bakarsayutisaman, Abu, and Issa Abdramane. "Solving a Reconfigurable Maze Using Hybrid Wall Follower Algorithm." International Journal of Computer Applications IJCA 82.3 (2013): 22-26. International Journal For Computer Applications. Nov. 2013. Web. 13 Nov. 2015. <http://research.ijcaonline.org/volume82/number3/pxc3892114.pdf>.

2. “Dead Reckoning.” Wikipedia: The Free Encyclopedia. Wikimedia Foundation, Inc. 12th November 2015. Web. 14th November 2015.< https://en.wikipedia.org/wiki/Dead_reckoning>

3. Hammond, Kris. "What Is Artificial Intelligence?" Computerworld. Computerworld, 10 Apr. 2015.

4. Myers, Jeff. "Statistics: Fastest-Growing Undergraduate STEM Degree | Amstat News." Statistics: Fastest-Growing Undergraduate STEM Degree | Amstat News. The Membership Magazine of the American Statistical Association, 1 Mar. 2015. Web. 11 Nov. 2015. <http://magazine.amstat.org/blog/2015/03/01/statistics-fastest-growing-undergraduate-stem-degree/>.

5. Nathaniel Johnston; et al. (21 August 2010). "Maze - LifeWiki". LifeWiki. Retrieved 1 March 2011.

6. Willardson, David M. "Analysis of Micromouse Maze Solving Algorithms." Analysis of Micromouse Maze Solving Algorithms (2001): n. pag. Portland State University Maseeh College of Engineering & Computer Science. Spring 2001. Web. 14 Nov. 2015. <http://web.cecs.pdx.edu/~edam/Reports/2001/DWillardson.pdf>.