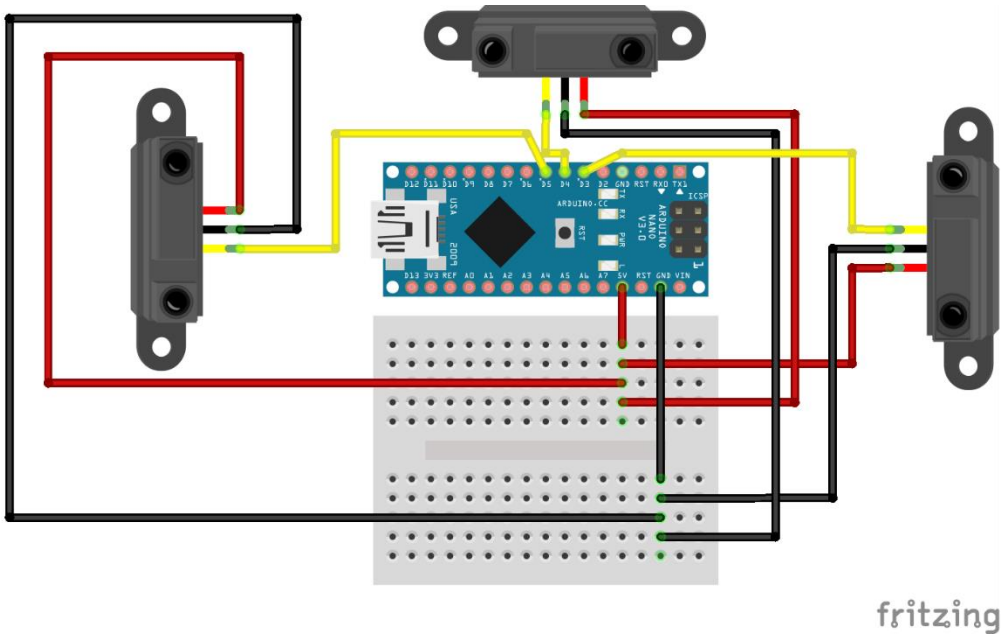
Harry in the maze

Part 1: The maze:

The circuit:



We used a simple circuit that utilizes 3 IR sensors to check whether the red line is there. All three sensors are connected to 5V VCC and ground. Each sensor is also connected to a pin on the Arduino to read the signal.

The code:

For this task, it was very important to note that this isn't straight forward and there are a lot of situations where a decision will have to be made on which direction to go. So a coordinate system was very important for this task.

```
7   class Coordinate{ // coordinate class to store x and y of a coordinate
8     public:
9     Coordinate(){
10    }
11    Coordinate(int x, int y){ // constructor to assign values
12      this->x = x;
13      this->y = y;
14    }
15    int x;
16    int y;
17  };
```

A simple coordinate class would be very helpful in storing the coordinates data.

```
39  int coordinates[] = {0,0}; // current coordinates
40  int sign[] = {1,1}; // sign of current coordinates (direction)
41  int i=1; // idicates whether we are moving on x or y
```

These three lines are very important in this task as they tell us which direction we are currently facing. The coordinates variable holds our current position in the maze. The sign variable holds our orientation. And the I variable holds the axis that we are going on. Using the sign and I variables together we are able to tell which direction we are heading and adding that to the coordinates implements a very strong direction system that can tell us where we currently are.

```
43  LinkedList<Coordinate> visited; // linked list to store visited coordinates in crossroads
```

Another very important aspect is that at a crossroad we must check each road to find the correct one, which could cause an infinite loop if done incorrectly. Here we have a linked list that stores previously visited coordinates. However, if we were to store every single coordinate, this could get very memory intensive. So, we only store one coordinate of a potential path at a crossroad.

```
19  // function to search through a linked list given the target coordinate
20  bool search(LinkedList<Coordinate> cont, Coordinate target){
21    for(int i=0;i<cont.size();i++){
22      Coordinate tmp = cont.get(i);
23      if(tmp.x==target.x && tmp.y == target.y)
24        return true;
25    }
26    return false;
27  }
```

This is a very simple linear search function that checks if a given coordinate has been visited yet.

```
52  void loop()
53  {
54    // while we only have forward direction, move forward
55    while(digitalRead(front)==HIGH && digitalRead(left)==LOW && digitalRead(right)==LOW){
56      moveForward();
57      coordinates[i] += sign[i]; // update current coordinates as you go
58    }
```

The first part of the loop is straightforward. As long as forward is the only direction to go, keep moving forward. Line 57 is updating the coordinates by the current direction where if we are moving on the x direction, I would be equal to 0, if we were moving in the y direction, I would be 1. If we were going in the positive direction, sing[i] would be 1 and if we were going in the negative direction, sign[i] would be -1.

```
59    // if we dont have any directions to go to go back
60    if(digitalRead(front)==LOW && digitalRead(left)==LOW && digitalRead(right)==LOW){
61      rotate90left();
62      rotate90left();
63      sign[i] *= -1;
64    }
```

If there is nowhere to go, simply turn back.

```
67    // flag to indicate that we have more than one road we can take
68    bool crossroad = digitalRead(front)==HIGH && digitalRead(left)==HIGH ||
69                     digitalRead(right)==HIGH && digitalRead(left)==HIGH ||
70                     digitalRead(front)==HIGH && digitalRead(right)==HIGH;
```

This Boolean checks if there are two or more roads available to take as it will be very important to check later on.

```
71    if(crossroad){
72      // add the road we just took to visited coordinates
73      int temp[] = {coordinates[0], coordinates[1]};
74      temp[i] -= sign[i];
75      Coordinate tmp = Coordinate(temp[0], temp[1]);
76      visited.add(tmp);
77    }
```

If we have more than one path to take, first add the path we just came from to the visited list so that in case we get back to this point again we know not to take this road again. Line 74 is adding the last point of the path instead of the crossroad point itself.

```
79    // if we are at a crossrads we check forward then left then right
80    if(digitalRead(front)==HIGH){
81      // we need to check if we already visited this so we check first coordinate of this road
82      int temp[] = {coordinates[0], coordinates[1]};
83      temp[i] += sign[i];
84      Coordinate tmp = Coordinate(temp[0], temp[1]);
85      if(!search(visited,tmp)){ // if not visited then move forward
86        moveForward();
87        coordinates[i] += sign[i];
88      }
89    }
```

In this case we don’t have to check if it’s a crossroad because if it wasn’t either the earlier loop wouldn’t have ended or this if condition wouldn’t be true. Before taking the road, we need to check if it has been visited before. We check the first coordinate of the path as that

would've been the one added to the visited list. If it hasn't been visited then we move forward.

```
91  if(digitalRead(left)==HIGH){
92    // temporary values to check direction before actually switching to it
93    int tmpi, tmpsign;
94    if(i){ // if we are going on y direction
95      tmpi = 0;
96      if(sign[i]){ // if we are going up
97        tmpsign = -1; // switch to left
98      }else{ // if we are going down
99        tmpsign = 1; // siwtch to right
100     }
101   }else{ // if we are going on x direction
102     tmpi = 1;
103     if(sign[i]) // if we are going right
104       tmpsign = 1; // switch to up
105     else // if we are going left
106       tmpsign = -1; // switch to down
107   }
```

In the case of the left direction being available we first need to parse the direction to know where we would be heading after turning left. We store it in temporary variables as the road might not be suitable so we don't need to change the main direction yet.

```
108   if(crossroad){ // if we have more than one option
109     // we need to check if we already visited this so we check first coordinate of this road
110     int temp[] = {coordinates[0], coordinates[1]};
111     temp[tmpi] += tmpsign;
112     Coordinate tmp = Coordinate(temp[0], temp[1]);
113     if(!search(visited,tmp)){ // if not visited then move forward
114       rotate90left();
115       i = tmpi;
116       sign[i] = tmpsign;
117       moveForward();
118       coordinates[i] += sign[i];
119     }
120   }else{
121     rotate90left();
122     i = tmpi;
123     sign[i] = tmpsign;
124     moveForward();
125     coordinates[i] += sign[i];
126   }
127 }
```

In the case of a crossroad, similar to the forward direction, we check the first coordinate of that road. If it hasn't been visited we change the

main direction and visit that road. If it isn't a crossroad, this is our only option so we just change direction without any other checks.

```
128   if(digitalRead(right)==HIGH){
129     // temporary values to check direction before actually switching to it
130     int tmpi, tmpsign;
131     if(i){ // if we are going on y direction
132       tmpi = 0;
133       if(sign[i]){ // if we are going up
134         tmpsign = 1; // switch to right
135       }else{ // if we are going down
136         tmpsign = -1; // siwtch to left
137       }
138     }else{ // if we are going on x direction
139       tmpi = 1;
140       if(sign[i]) // if we are going right
141         tmpsign = -1; // switch to down
142       else // if we are going left
143         tmpsign = 1; // switch to up
144     }
145     if(crossroad){ // if we have more than one option
146       // we need to check if we already visited this so we check first coordinate of this road
147       int temp[] = {coordinates[0], coordinates[1]};
148       temp[tmpi] += tmpsign;
149       Coordinate tmp = Coordinate(temp[0], temp[1]);
150       if(!search(visited,tmp)){ // if not visited then move forward
151         rotate90right();
152         i = tmpi;
153         sign[i] = tmpsign;
154         moveForward();
155         coordinates[i] += sign[i];
156       }
```

```
156         }
157       }else{
158         rotate90right();
159         i = tmpi;
160         sign[i] = tmpsign;
161         moveForward();
162         coordinates[i] += sign[i];
163       }
164     }
165
166   }
```

The right direction is the exact same as the left direction with simple changes in the rotate functions.

Part 2: The safe:

```
#define PPR 2048

// encoder pins
#define B1 2
#define A1 3

#define B2 18
#define A2 19

#define B3 20
#define A3 21

#define led 8
```

Some definitions used for easy access. Each of these encoder pins are defined to interrupt pins which is the reason this task cannot be done without the use of an Arduino mega with 6 interrupt pins.

```
int counter[]= {0, 0, 0}; // counters for encoders
```

A counter array that holds a counter for each encoder.

```
// interrupt functions that calulate encoder movement
void ISR_A1(){
  if(digitalRead(A1)!= digitalRead(B1)){
    counter[0]++;
  }else{
    counter[0]--;
  }
}
void ISR_B1(){
  if(digitalRead(A1)!= digitalRead(B1)){
    counter[0]--;
  }else{
    counter[0]++;
  }
}
```

Interrupt functions that adjust the counter according to the direction of movement. This has been done for each of the encoders.

```
63   void setup() {
64     // setting pin modes and attaching interrupts to encoders
65     pinMode(A1, INPUT_PULLUP);
66     pinMode(B1, INPUT_PULLUP);
67     attachInterrupt(digitalPinToInterrupt(A1), ISR_A1, CHANGE);
68     attachInterrupt(digitalPinToInterrupt(B1), ISR_B1, CHANGE);
69
70     pinMode(A2, INPUT_PULLUP);
71     pinMode(B2, INPUT_PULLUP);
72     attachInterrupt(digitalPinToInterrupt(A2), ISR_A2, CHANGE);
73     attachInterrupt(digitalPinToInterrupt(B2), ISR_B2, CHANGE);
74
75     pinMode(A3, INPUT_PULLUP);
76     pinMode(B3, INPUT_PULLUP);
77     attachInterrupt(digitalPinToInterrupt(A3), ISR_A3, CHANGE);
78     attachInterrupt(digitalPinToInterrupt(B3), ISR_B3, CHANGE);
79
80     pinMode(led, OUTPUT);
81
82   }
```

Setting up the pin modes as well as attaching the interrupts to the encoder pins.

```
84   void loop() {
85     // calculating degrees moved relative to start
86     int degree1 = counter[0] / 4 / PPR * 360;
87     int degree2 = counter[1] / 4 / PPR * 360;
88     int degree3 = counter[2] / 4 / PPR * 360;
89
90     if(degree1 == 37 && degree2 == 10 && degree3 == 54){ // if the correct degrees are set
91       digitalWrite(led, HIGH); // turn on led
92     }else{
93       digitalWrite(led, LOW);// else turn off led
94     }
95
96   }
```

In the loop, for each of the encoders, we calculate the relative degree since the start of the program by dividing the count by 4 then dividing it by the selected PPR then multiplying by 360 to get the degrees moved.

Whenever that degree reaches the correct number for each encoder, turn on the green led, otherwise keep it off.