SearchAndRescue.ino   
Contains all our functions that we call to allow us to move around the map. This includes forward and backward movement, rotating or scanning rooms. It also has several switch statements that it uses to decide which movement to do based on the user input.

The two biggest issues I had with the project was the forward movement due to the motor’s power being out of sync, one was more powerful than the other. This meant that it would curve into a wall and wouldn’t go straight. The second was the turning not being precise, it took me a week to finally come to a realisation that it’s impossible with the hardware we had to get it perfect all the time. In this part it explains the steps I took to get the forward movement working and how I got the rotations working.   
  
P.S – In the READ\_ME I’m only discussing here the different approaches I took, where I found them and the ones I have settled on.   
  
I don’t believe I need to talk about my code here. This can be seen within the actual code itself since the naming of the variables explain what each one is used for and part of the code that don’t have naming have comments explaining what it does. Again, this is only discussing the approaches I took, where I found them and the ones I have settled on.

## Moving forward

I originally used just the motor speeds and delays to go across the map this worked but was inconsistent and was prone to getting stuck on walls or being stuck in the corner of a corridor when used with the line sensors. It was also not very accurate, so I had to find another alternative.

I tried numerous attempts in hardcoding the motor that had less power however again it was unreliable. I came across another method that adjusted the speed according to what angle we are currently, we would get this angle from the gyro. <https://github.com/pvcraven/zumo_32u4_examples/blob/master/HeadingHold/HeadingHold.ino>  
More information on it here. However, it was still slightly curving, and I wanted to be as straight as it can be so I looked again to see what I could find on the web.

I found a method that used the encoders and it was this method that I have used in all the movement and rotation functions. Using the encoders to work out the correction, depending which one you want to compensate for. Example if it's the right motor. You would multiply the right encoder value with a number between 0.9 to 1.1, this is to adjust for any minor curves, I believe mine is 1.015. You then take this value off the left encoder to give us our correction, you then add this correction to the current speed of the motors. Then add the result to the speed of the right motor. This will apply the correct compensation, either adding or deducting speed away to keep us in a straight line. This has worked very well for me and so I've used it.  
http://www.abrowndesign.com/2017/02/25/zumo-32u4-synchronize-motor/  
This was the handy guide that helped me correct my motor issues. I applied this also to my rotations which works most the time. I believe the reason the rotations aren’t always accurate is due to the gyro.

## Line Sensors

I also had to make sure the Zumo would stop when it hits the wall this was when we used the line sensors to detect if it was over a line or not. Luckily, I didn’t have to much of an issue with this thanks to following the Maze Solver in the zumo32u4 examples. This allowed us to calibrate the line sensors allowing for more precision for the sensors when it goes over a line. I did have to add a delay though once it hit a right or left sensor since the middle sensor is further back, so it needs time to get into position. This allows us to know if we hit a wall straight or if we hit the wall at an angle.

The TurnSensor.cpp and TurnSensor.h was taken from the Maze Solver example. This allows us to calibrate, update and reset are gyro. It also converts it into an angle which we can use to rotate our Zumo. This was necessary for the calibrate line sensors where it had to rotate left and right.

## Rotations

Once I realised that it wasn’t possible to get the turning 100% accurate all the time the Rotations was fairly straight forward I used the Gyro to work out what angle I am at, I needed to make sure we updated the angle each loop so the gyro knows how much it has turned by. You then want to make sure you reset the gyro back to zero once you're finished or your gyro data may become invalid. Only do this on any rotations and not moving forward or reversing etc. it messes up the values.  
<https://github.com/pvcraven/zumo_32u4_examples/blob/master/TurnExample/TurnExample.ino>

^I used this example in my code, I did use the Maze Solver example but found this one better.

I had tested rotating by just using encoders which is commented out in my code somewhere and delays but again the same issue they we're not accurate enough and the Gyro was the one that proved more reliable but still isn't 100%. I applied the encoder speed correction to the Gyro to compensate for the power difference in the motors that had improved the accuracy by a small margin.

## Proximity Sensors

The proximity sensors were probably the simplest to get working, this allows us to know if there is an object that has been detected in a room. I had to play around with the Proximity Sensor Threshold, but I found 5 - 6 was the best. I used the SumoProximitySensors Zumo32u4 example to give me an understanding of how to apply it and was able to get it working fine on the first try.

## BuzzerAndLED.h and BuzzerAndLED.cpp

Contains the functions that allow our buzzer to play a tune and allows us to have turn on or off our LED lights on top of making them flash I was able to understand how to use the LED lights by following the example BlinkLED and I looked at the class reference for the buzzer it is also where I got my tune from. https://pololu.github.io/zumo-32u4-arduino library/class\_zumo32\_u4\_buzzer.html#ab72bde97ceceef8705f1bbaeccb970db

## SerialMessages.h and SerialMessages.cpp

This is where we store the messages for our code. I realised in my code that the messages are repeated a lot, so I condensed them down to save memory. (Dynamic Memory is now down to 54%. it originally 86%, was a good choice doing this). Remember and this is very important, Serial1 is what we use to communicate to the XBEE and Serial is what we use to communicate through the USB cable.

MotorSpeedAndDuration.h and MotorSpeedAndDuration.cpp   
Provides the variable speeds for the motors, sets the time limit for any delays and provides functions that set the motor speeds. I investigated the class reference mo32u4 Zumo motors to understand how it all works and helped by the Maze Solver example. More information in the code but this explains what this class and header does.

## TurnSensor.h and TurnSensor.cpp

This was taken from the Maze Solver example. This was required to allow us to calibrate the Gyro as well as use functions such as updating the Gyro or resetting the Gyro back to zero. It also gave us the angle in which are Gyro is currently at and is very important in getting the Gyro functioning correctly or at least to a decent standard since it’s not always perfect.

## Oldcode.cpp

Holds all our old cold that is not used anymore and comments explaining why it was taken out.

## GUI

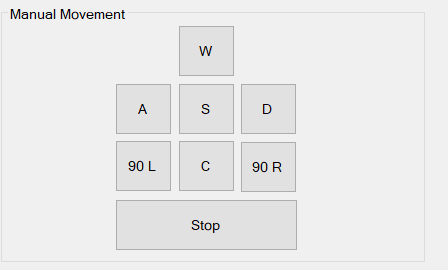
The most difficult part about the GUI was figuring out how to get the GUI to read incoming and sending messages from the serial. Luckily, I found a very basic tutorial that allowed me to create a very simple version. There was a more advanced version that allowed you to check every COM port to see if there was one in a dropdown and would message you if you were connected but for what we needed I went simple instead of over engineering it.

<https://www.instructables.com/id/C-Serial-Communication-With-Arduino/>  
^ the example I followed.

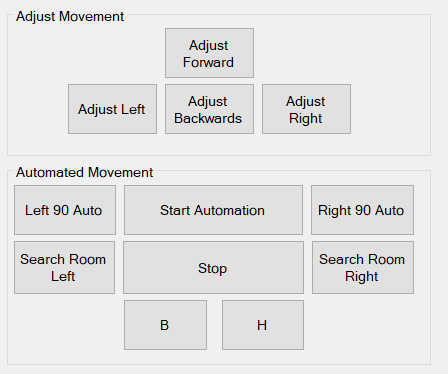
I used COM6 on my laptop to read incoming messages from the XBEE, I also use windows forms to create my GUI and since I’ve used it before it was straight forward. To read messages I use a stop timer that gets turned on when I connect to the serial port. The timer will read incoming existed messages and is filtered so that’s how I get messages on text box.

## Task 1, 2 and 3

The first part before even using the GUI is you must press the A button on the Zumo to calibrate its Gyro and the line sensors.

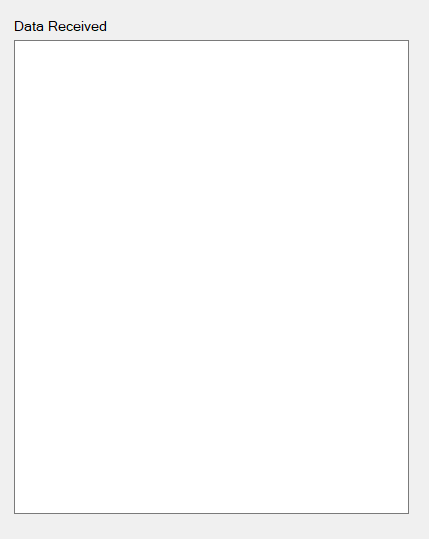
  
This is the manual control of our GUI for the manual movement. If you click W this will allow you to move forward, when you click A this will make the Zumo rotate left, when you click D this will rotate the Zumo right and when you press S this will reverse. Now these movements will carry on until you tell it to stop moving by using the stop button.   
  
You can click C which will automate the Zumo to go forward. If the Zumo hits a wall at an angle the left or right line sensor will detect this and will apply the correction algorithm in the code to get it back on track. If it hits a wall head on is when either the left or right sensor has been hit but also the middle sensor has been hit. This will then stop the robot and allow the user to put in a input.  
  
You can click 90 L to rotate 90 degrees to the left or use 90 R to rotate 90 degrees to the right.  
  
P.S – You can not use manual movement if you want to return home (Task 7) the reason for this is that I don’t read the encoder values when manually moving since I would be doing it automated. Only use manual movement when you are NOT going to use (task 7) or you would need to reset.

## Task 2, 3, 4, 5, 6, 7



The first part before even using the GUI is you must press the A button on the Zumo to calibrate its Gyro and the line sensors.

The Adjust movement controls are mainly for if the Zumo has not turned correctly or it’s managed to get stuck in an awkward spot due to the map. I was told we could use this, so I decided to add it in. This can only be used when the Zumo has stopped fully, you can also stop the Zumo yourself if you want to correct its trajectory slightly. Clicking one of these buttons will move the Zumo slightly depending on the one you’ve clicked.

The Automated Movement Controls has all task 2,3,4,5,6,7 in them. Start Automation is exactly what clicking C does in Manual Control. Left 90 Auto and Right 90 Auto allows us to turn a corner and carry on moving forward without having to tell it to.   
  
  
Task 5

You must make sure to stop when you want to use search room left or search room right. You can’t click them while the Zumo is moving the same is for every button to be honest. These search buttons allow us to go into a room and scan the room using the proximity sensors. It will then throw a message on the data input box letting us know the room number and if there was survivor in there or not.  
  
Task6

Pressing B is done when you are at a junction and you’ve chosen either the left or right corridor to go down. Once you’ve reached the end of that corridor is when you click B. This will turn you 180 degrees; it will then put you in a switch that only allows you to adjust the position of the Zumo in case it’s managed to get itself into a funny position or carry on moving forward. No inputs can be made to the Zumo till is has reached the T junction.  
  
Task7  
Once you are done exploring the map, you then click H to return home this will put the Zumo into Auto control. It will only search places where survivors were found and has an optimized path that ignore corridors that had no survivors in them. It will scan the room again that originally had a survivor in and let you know on the user input if there was a survivor inside or if it has left. It will flash LED lights and play a tune if there is a survivor inside. Once Home the LED lights should switch off.

Each distance covered is stored in the array and then the movement initiated is stored in the array. Then we press H to go back, it will then go in reverse and move forward, till it hits the stored encoder values. Then do the movement which would either be a rotation or search room. This will keep happening till we return home.