Alastair Macmillan
Ollie Whiteman
Tom Newman

COSC343 ASSIGNMENT 1 GROUP REPORT

We have designed an algorithm written in python that moves an EV3 mindstorm from a starting tile towards a track of small black and white tiles. After turning 90 degrees clockwise it then proceeds to move down this track of tiles counting 15 black tiles which are signified with a beeping sound as it passes over them. It then turns 90 degrees and moves towards a tower which it senses and pushes off its resting position on a black tile.

Setup:
For this piece of code we used the MoveTank(), ColorSensor(), TouchSensor(), Sound() and UltrasonicSensor() methods. We then set global variables for the reflected light intensities we predict the different tiles to be. We created utility methods to the specific levels of white and black, with grey being between those values.

Move Onto Track:
Initially the mindstorm starts on a large black tile perpendicular to the track of small black and white tiles. We set both motors to move at the same speed relatively slowly. While the motors are running, it waits until it senses grey, then waits until it senses black. This is so that it passes the grey tile and senses the black tile to start the path on. It then waits a small amount of time so that the bot is lined up in the middle of the tile, then rotates to align with the track.

Recolor: (unused)
The recolor subroutine was a small piece of code that would have run after move_onto_track. It would have sensed the light intensity of the black tile, and changed all light variables to match. This, in theory, should account for different lighting conditions. However, in practice, the program confused black for grey and vice versa, which is why during final testing we decided not to use this method and instead set static values for the light intensities, as this approach provided far greater consistency.

Counting Black Tiles Algorithm:
This code uses a lot of state variables. In order to count the black tiles correctly, the robot instead has to wait until it has passed a white tile. If it just counted the number of times it saw a black title naively, it would count the same tile 15 times.

While the move_onto_track algorithm is good at aligning the robot, neither the program nor the robot can be that precise. So, occasionally, it drifts onto the grey tiles. When it detects grey, it moves forward slightly, to make sure that it is not on a seam, and then tries again. If it is still on a grey tile it pans left and right to find a black or white tile, it then moves in the direction that it detected either a black or white tile. It then moves forward onto the tile. However, in this state, it is on a weird angle due to the fact it panned left or right to get to this state. So, remembering the way it turned previously. It turns the opposite direction to compensate, but slightly less, so it is less likely to bump into that particular wall. Due to the distance it moved forward during the correction procedure, it might have skipped a tile. So, in order to compensate, the robot then moves backwards slightly, then continues counting tiles.

Alastair Macmillan
Ollie Whiteman
Tom Newman

Finding Tower Algorithm:
Assuming that the realignment method during part 1 (counting black tiles) worked, the robot should be facing roughly correctly. So it makes a 90 degree turn and blindly drives about two thirds of the way to the tower. Although this is not necessarily accurate, it allows us to easily get within range of the mindstorm's ultrasonic sensor, and to begin trying to sense the exact course it must take to get to the tower. Once it has moved within range of the ultrasonic sensor it begins a sweeping behaviour to scan and accurately steer towards the tower.
When executing the sweep, it first rotates 90 degrees to its left on the spot, then begins rotating 180 degrees to the right until the ultrasonic detects the tower. However the sensor will detect the edge of the tower. In order to compensate, we have added a small sleep time in order for the mindstorm to keep rotating a little and square up with the middle of the tower. It now begins its course to the tower driving three rotations before repeating the sweep again.
The touch sensor is then used to check whether contact has been made with the tower, if contact has been made it increases its motor speed to thrust the milk bottle off the black tile making four rotations before making a beep sound to signify completion of the task.

Problems Encountered:
A major stumbling block we found when writing the algorithm was finding the correct reflected light intensity values to use to specify the different colours of the lobby terrain. Under perfect conditions our realignment method worked perfectly however the reflected light intensities were found to be extremely variable making for an agonizing task of fine tuning.
When writing the Finding Tower Algorithm we used a trial and error approach to adjust the input variables for the different functionality of the mindstorm. This was time consuming as we had to walk back and forth from the lobby to the lab making fine tune adjustments. We had to manually adjust the distance it initially drove to the tower, the sensitivity of the sleep after detecting the tower in order to square up to it and the amount of sweeps done to ensure an accurate path was taken.
By far the hardest problem we encountered was fine tuning the realignment functionality, finding out a way to get back on track, re-align straight and not miss a black tile took a lot of different ideas and testing. We came up with many ideas such as reversing back, trying to find the edges of the tiles to know where not to drive, and many others. In the end we decided to pan the robot to find the course once it was on a grey, go towards that direction to get back on track. The re-aligning method as described in the algorithms section earlier was a very tedious task, trying to figure out an idea of how to straighten up and then implement it proved to be time consuming. Manually adjusting the smallest of turns and then testing over and over was the reason that this was the hardest and most time consuming problem we had to fix.