Group 4 - 20
Waterloo Engineering Expeller of Dominoes

Department of Mechanical and Mechatronics Engineering

MTE 100 / MTE 121

Prepared by:

Josh Morcombe - 20937588

Andor Siegers - 20990622

Henrique Rodrigues - 21037291

Sean Aitken - 21006546

Tuesday, 6 December 2022

Table Of Contents

List Of Figures

List Of Tables

Acknowledgements

The group would like to acknowledge the company that manufactured the dominoes that were used in this project, Spin Master Games, as well as Xander Soldaat for creating a RobotC library for the EV3 multiplexer. They would also like to acknowledge the incredible Instructors and TA’s that made this project possible with their knowledge and dedication.

Summary

The task of setting up dominoes to be knocked down in a chain reaction is tedious and time consuming. To solve this issue the Waterloo Engineering Expeller of Dominoes was created. This robot was able to set up 30 dominoes in a line while following a list of instructions from a file or by following a line drawn on the ground.

The project was broken into smaller tasks for each group member such as building individual mechanical subsystems or programming functions. A schedule was also created and revised to ensure a timely completion of the project.

To build this robot, Lego pieces were used to construct a physical assembly which integrated sensors and motors from the EV3 kit. The robot was designed to move dominoes from a storage hopper down a release chute through the use of an actuated dispenser arm. These dominoes would then fall to land in a standing position. A rear door would then open, leaving the domino standing on its own. These subsystems were integrated together and programmed using RobotC to leave a free standing line of dominoes behind the robot.

A digital path was created using a Python program, which integrated the pygame library to create instructions for the robot to follow. The goal of the robot was to set up dominoes in a line while following a digital path or drawn line while responding to external factors. Some of these factors included objects placed in front of the robot, which prompted the robot to pause operation until the obstacle was removed, or user inputs to the touch sensor which activated the shutdown procedure. This shutdown procedure would end the setting up of dominoes either once the robot ran out of dominoes or the touch sensor was pressed.

The goal of setting up 30 dominoes in a timely manner was met, as the robot could set up all dominoes in under five minutes. The robot was also able to successfully follow the digital path or drawn line. In the end, the Waterloo Engineering Expeller of Dominoes was designed and built within the scheduled time and was able to set up dominoes reliably through a strong mechanical design and effective programming. The specifics of the mechanical and software design of the Waterloo Engineering Expeller of Dominoes, as well as ideas for future improvements are outlined below.

1.0 Introduction

Creating a standing line of dominoes is done for the purpose of knocking them all down in a large chain reaction. However, this goal can be tedious or inaccessible to some. The coordination and spacing required to set up dominoes can sometimes be demanding causing the entire chain to fall. This coordination and repetition of tasks is exactly what a robot can be used to complete. The Waterloo Engineering Expeller of Dominoes was created to solve this exact problem. Automating the process of placing dominoes to create a robot that not only sets the dominoes up in a straight and consistent fashion, but also places them along a simple to create, predetermined path.

2.0 Scope

2.1 Main Functionality

The robot completed the task of setting up dominoes automatically in a path that can be knocked over in one chain reaction. The path that it laid the dominoes on was determined either by a line that was drawn on the ground that the robot followed through colour sensors, or by a path that was drawn digitally and sent to a file which the robot read. It then proceeds along the path, cyclically dispensing dominoes out behind it. Functionality was added to automatically pause all operations if an obstacle was placed in front of it while simultaneously alerting the user with a sound. It would then proceed once the obstacle was removed. Finally, the robot was programmed to have the option to knock down the domino chain on its own, or wait for the user to knock them down themselves.

2.2 Inputs

The robot had a number of inputs that it used in its regular operation. Firstly, it took information from the buttons on the EV3 Brick in order to select an operating mode. In file follow mode, it would receive a file of instructions (previously a list of coordinates) that was generated beforehand and drive the appropriate path. In line follow mode, the robot would detect a coloured line below it and follow this line, placing dominoes. Initially this was planned to be done with a single colour sensor and gyro, however a method using two colour sensors was more favourable. It would use information received from two colour sensors placed on the bottom of the robot to follow a line drawn on the ground. Regardless of the mode, it would continuously read the inputs from the ultrasonic sensor to detect obstacles or read from the touch sensor to start the shutdown procedure. This was changed from the initial idea of the touch sensor functioning as an emergency stop button.

2.3 Interaction with the Environment

The robot had four motors that allowed it to interact with its environment. Firstly, two large motors were connected to the wheels, driving the chassis. These allowed the robot to move across the ground in two dimensions and thereby complete its designated path. There was also one medium motor connected to a dispensing arm that pushed dominoes from the domino hopper down a ramp and into its final position. Lastly, another medium motor operated the door, allowing the dominoes to be released out the back of the robot. The colour sensors on the robot were used to follow the line by detecting the white space on either side of the drawn line. If either sensor did detect the line below, it would change the robot's path accordingly to stay directly on top of the line. As mentioned previously the ultrasonic sensor would sense for obstacles in the way and alert the user to remove it. The rest of the previous inputs described were all human controlled.

2.4 Shutdown Procedure

The general operation of the robot consisted of one of two main operational loops depending on which mode was selected. One way to exit these loops was for the robot to run out of dominoes; there was an internal count stored as an integer variable that would decrement each time a domino was released, and the robot would enter its shutdown procedure once this value reached zero. Alternatively, the loop would be exited if the touch sensor was triggered, at which point the robot would automatically topple the last domino placed and shutdown.

To begin the shutdown procedure, the robot would first stop all operations, that is, all motors would be turned off and all loops would be exited. The robot would then wait for the touch sensor to be pressed which would cause it to topple the first domino starting a chain reaction. If the touch sensor was not pressed within ten seconds of the start of the end procedure, the robot would exit the program without toppling the dominoes.

2.5 Changes to the Scope

The largest change that was made to the scope of the project since the beginning was the decision to use two colour sensors to follow the line instead of one. While the initial intention was to use just one colour sensor working in tandem with a gyro sensor, it was decided that it would be very difficult if not impossible to create a reliable path-following program with this combination of sensors. The plan was then altered accordingly to use two colour sensors, making it more realistic and achievable.

Furthermore, a small alteration was also made to the functionality of the touch sensor. As mentioned above, the touch sensor was originally going to serve the purpose of an emergency stop button. It was decided that it would be more useful if it also made the robot topple the first domino, as it added extra functionality and was simple to implement.

3.0 Constraints and Criteria

3.1 Constraints

The robot had two main constraints: the amount of dominoes it could carry and the time it took to place them. The robot could carry up to 30 dominoes and had to place them in less than 5 minutes, since that was the maximum demonstration time. Both constraints were valuable in creating a realistic, achievable goal.

Over the course of the project, those constraints were changed. Initially the maximum number of dominoes to be carried was 60, because that was the available resources. However, it proved very difficult to carry that many dominoes because of the weight it created on top of the other dominoes, which increased friction. Secondly, the initial target-time planned for was ten minutes, however this was later cut to five due to demonstration time constraints.

3.2 Criteria

The robot was expected to place the dominoes correctly and evenly spaced from each other, and have them fall in a chain reaction when one was knocked over. It was also expected to follow a line or a path from a file uploaded to it while simultaneously setting the dominoes. The criteria were not changed during the development of the project. Both criteria were helpful in designing a functioning robot as they were crucial in guiding both the software and hardware design processes.

4.0 Mechanical Design and Implementation

The Waterloo Engineering Expeller of Dominoes was designed to efficiently set up dominoes along a created path which could then be toppled in a chain reaction. This was done by moving a domino from a storage hopper to the release chute through the use of an actuated arm. Once the domino fell down the release chute and landed standing up, the robot opened the release door and drove forward to leave the domino standing. The robot also utilised colour sensors to follow a line drawn on the ground or gyro sensors which allowed it to follow premade instructions. The robot also had functionality to react to external factors such as objects in its path or an input from the user through the touch sensor to knock over the chain of dominoes.

4.1 Chassis design

The chassis of the robot was designed with a focus on structural stability, connecting to all essential parts. These parts included the ultrasonic, colour, touch, and gyro sensors along with, the hopper, door assembly, drive motors, dispenser arm, release chute, and the Brick. The overall shape resulted in a drivetrain base with a sturdy frame extending to the front securing the dispenser arm as seen in *Figure 1*.

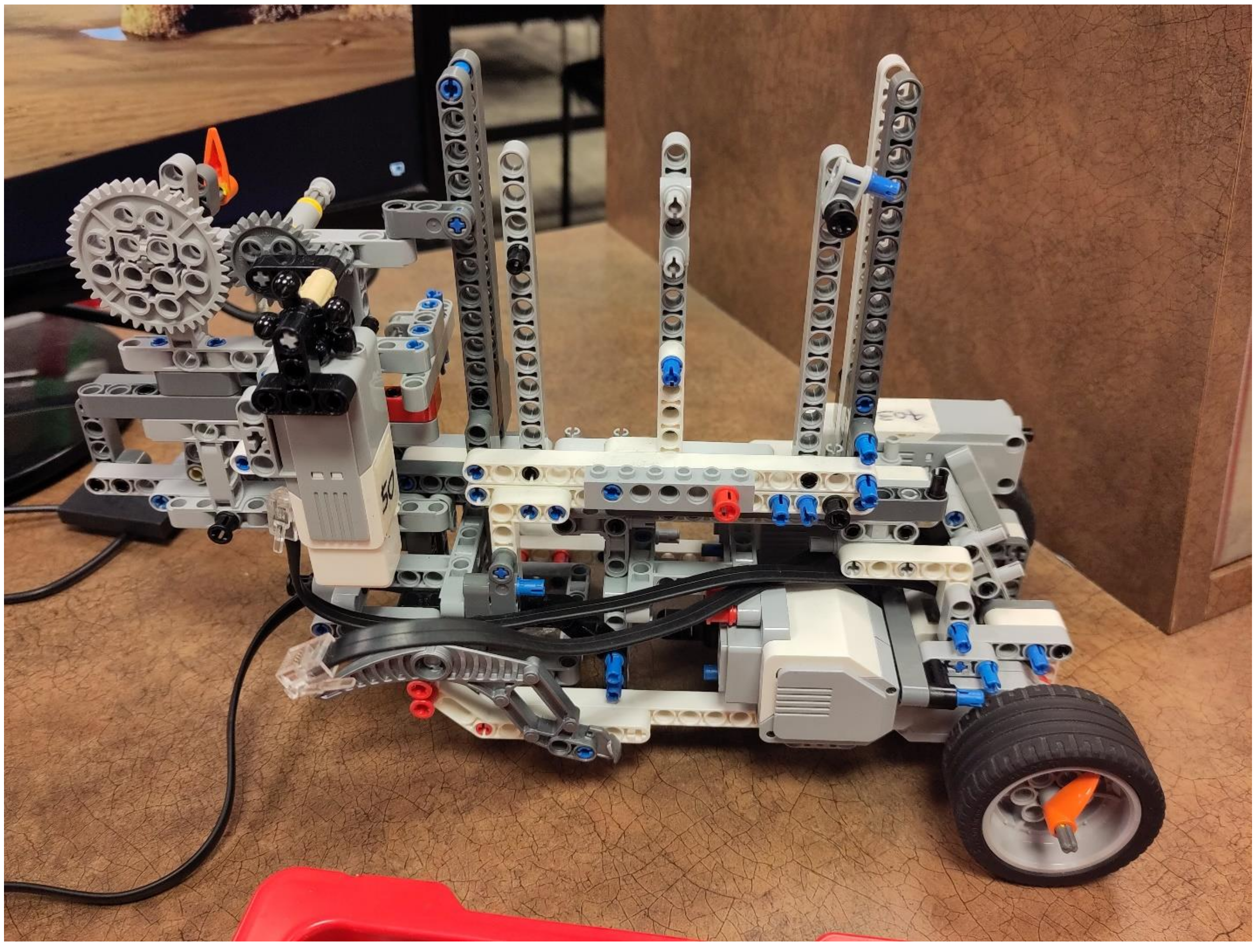

Figure 1: Chassis

<u>4.11 Hopper</u>

The hopper assembly was designed to allow for gravity to feed domino's down towards the dispenser arm. The original design consisted of three columns which held dominoes perpendicular to the release chute. The dominoes were to be dispensed and rotated 90 degrees as they fell into the correct orientation. After much testing, this design was determined to be too unreliable and was changed to a hopper with two columns in line with the release chute. This design held fewer dominoes but the path the dominoes took to the release chute was much more reliable. As seen in *Figure 2*, both columns of dominoes would fall to be in line with the dispenser arm, with the rest of the stored dominoes being stacked on top. As the dispenser arm contacted the first domino, it would connect with the second domino, pushing both dominoes. As the arm pushed the dominoes, it would slide under the stacked dominoes holding them up. Once both of the dominoes were dispensed, the arm would slide back from under the stacks of dominoes causing them to fall into place. This cycle could continue indefinitely until the robot ran out of dominoes.

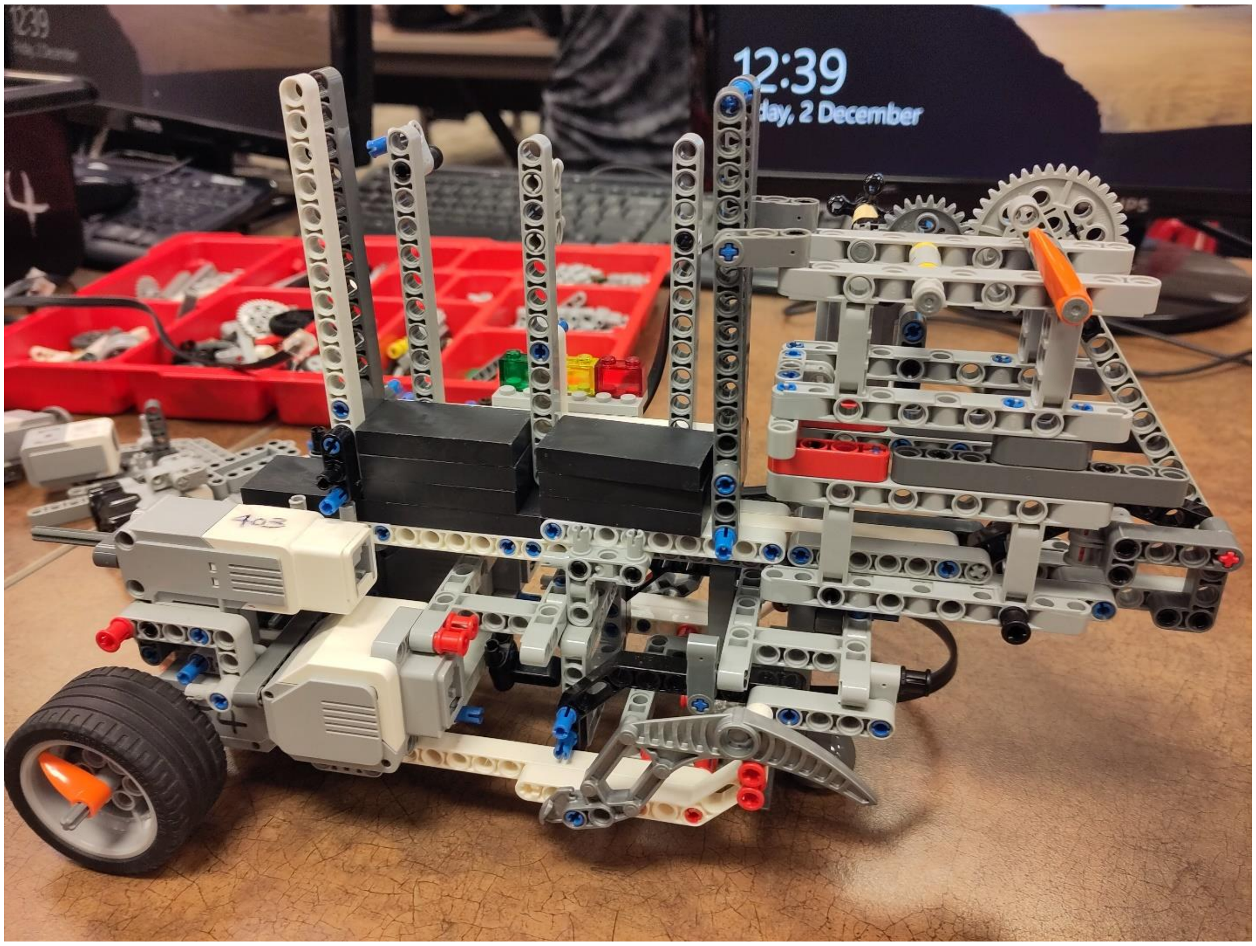

Figure 2: Hopper Section View with Dispenser Arm

<u>4.12 Release Chute</u>

The release chute was positioned at the back of the robot to funnel dominoes from the hopper to the release door. The chute was originally designed to orient dominoes the correct way before they fell into the release chamber. This proved to be difficult to do with Lego parts as the domino would need to rotate 90 degrees on two axes from the hopper to the release chamber. The final release chute was designed to be a simple ramp to guide a horizontal domino to a vertical one as seen in *Figure 3*.

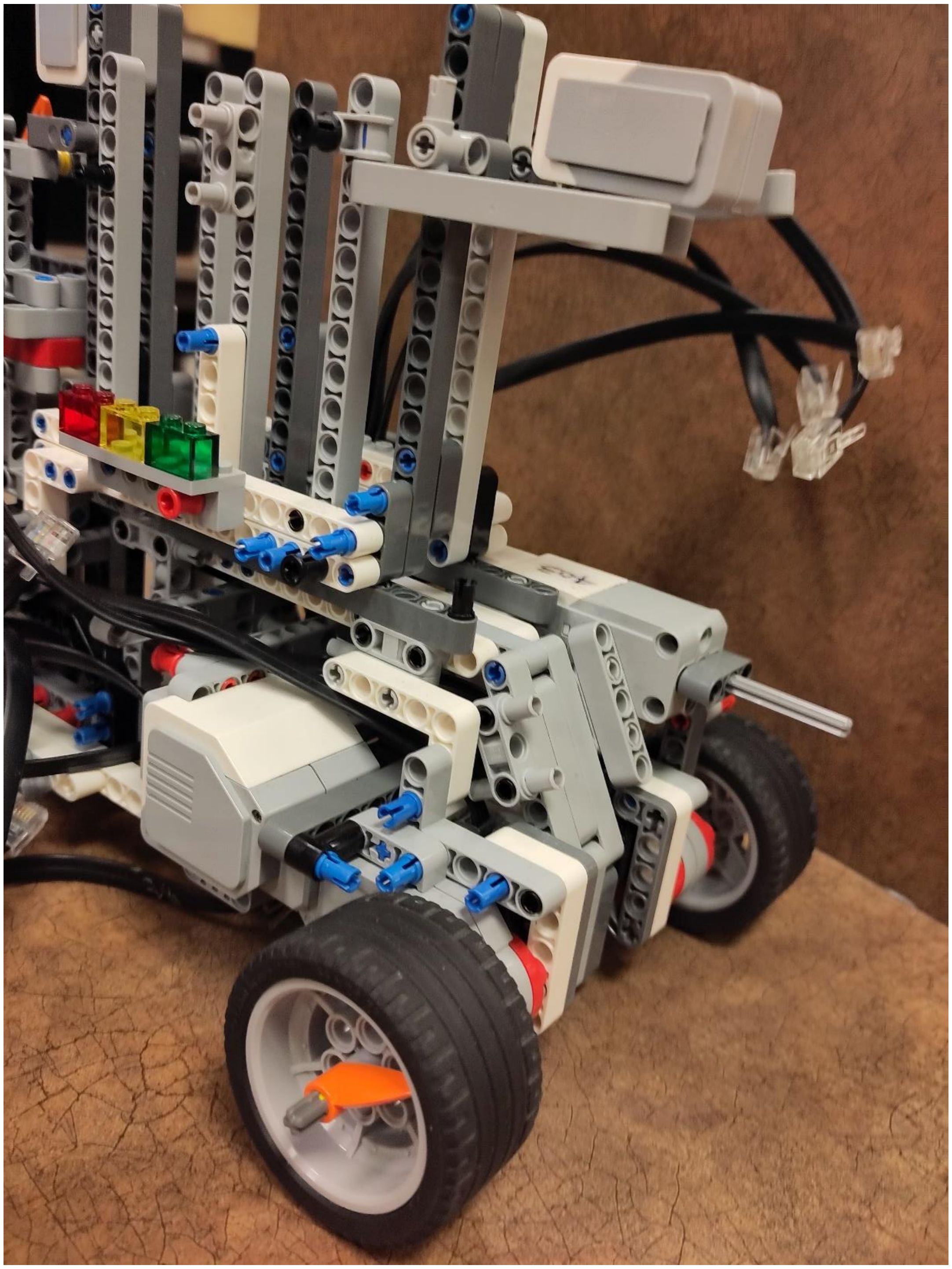
Figure 3: Release Chute Side View

4.2 Motor Drive Design

4.21 Drivetrain

The drivetrain assembly was constructed in a relatively simple fashion with a large motor powering the left wheel and a large motor powering the right wheel. A steel ball caster added a third point of contact with the ground which improved stability. Originally the motors drove the robot from the front, but this caused the back portion of the robot to swing out around turns. This was a critical flaw as dominoes were dispensed from the back and had to be positioned precisely on the line. The chassis was adjusted to have the driving motors be at the back, in line with where dominoes were dispensed. The position of the steel ball caster was also moved to the front. Consequently, the large motors had to be mounted and programmed backwards so they would fit the form of the chassis. The overall chassis as seen in *Figure 1*: Chassis was now driven from the back to allow for the dominoes to be accurately placed on the line.

4.22 Dispenser Arm

The dispenser arm was originally designed to resemble a piston and crankshaft with a rotating arm that actuated the dispenser arm in and out. This motion pushed a domino out then retracted to allow another to fall from the hopper. This design was later changed as the ratio of rotational to linear motion deviated too much. The newer design, as seen in *Figure 4*: Dispenser Arm was composed of a rotating arm, a horizontal moving pusher and a linkage bar. The lengths of the rotating arm and linkage arm were tested and rebuilt to find a combination that optimised the linear motion of the horizontal arm. The ratio of rotation of the rotating arm to the horizontal distance the pusher arm slid was then also more consistent. This design was slightly modified one more time as it was observed the rotating arm initially pushed down vertically on the assembly rather than horizontally. The length of the linkage and rotating arms were once again modified resulting in the most effective pushing arm.

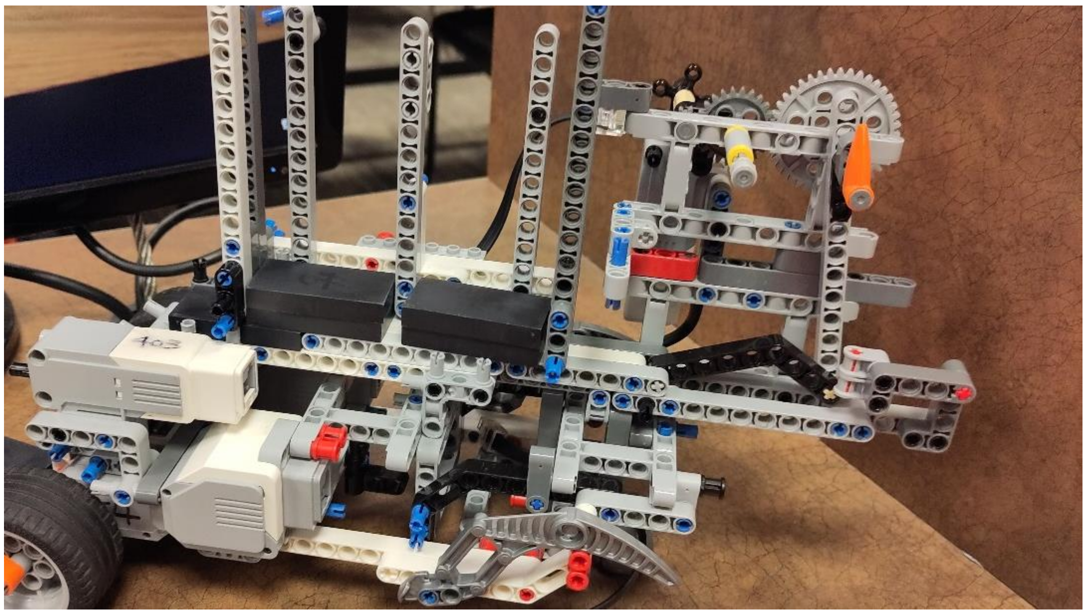

Figure 4: Dispenser Arm

4.23 Door

The rear door of the robot was designed so that the dominoes were stabilised before being released. The dominoes fell into a chamber between the chassis and the door, then the door opened and the whole robot drove forward leaving the domino standing. The robot then closed the door and repeated this process. The original door design consisted of a rack and pinion gear which rotated to bring the door upward. After several attempts at designing such a mechanism, it was found the gears would sometimes slip and the construction of the frame was overly complex. It was decided that a rotating door as seen in *Figure 5*: Back Door was much simpler and more reliable.

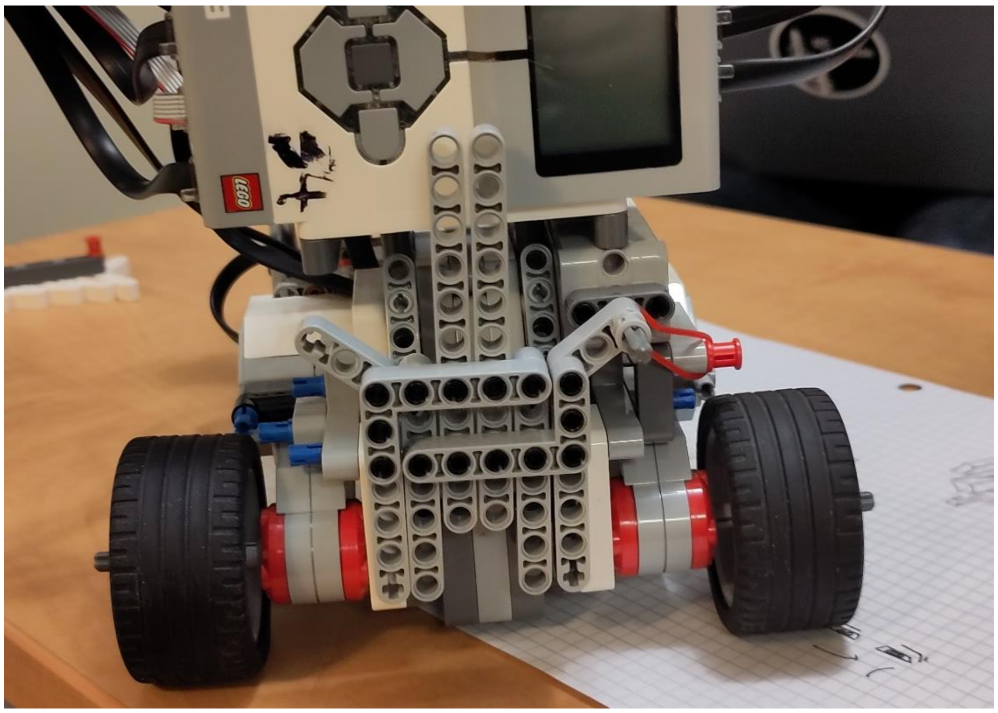

Figure 5: Back Door

4.3 Sensor Attachment Design

4.31 Color Sensors

A system of two colour sensors were used to accurately and consistently follow a black line. Both sensors were positioned to be in line with the axles of the drive wheels. This was because the robot had to follow the line, while still keeping the domino release area on the line. The sensors were mounted facing downward as seen in *Figure 6*: Robot Bottom View with the spacing between them being around the thickness of the drawn line.

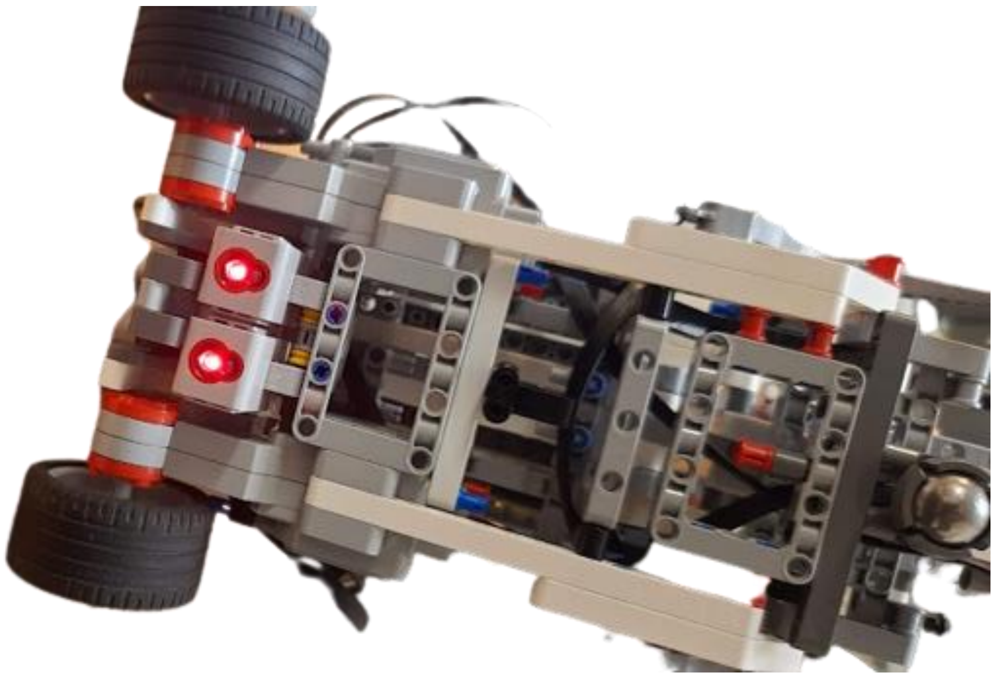

Figure 6: Robot Bottom View

4.32 Ultrasonic Sensor

The ultrasonic sensor was mounted on the front of the robot, facing forward, as seen in *Figure 7*: Ultrasonic Sensor. This allowed the robot to detect any objects and alert the user if there was something in the way of its path. This was done by stopping the robot, playing a sound, and displaying a message to the screen, until the object was removed.

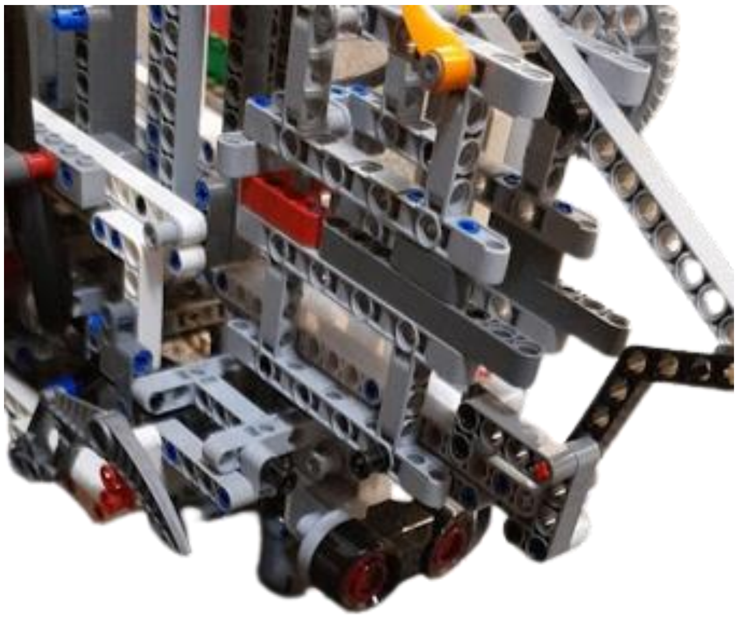

Figure 7: Ultrasonic Sensor

4.33 Touch Sensor

The touch sensor was positioned on top of the robot to be accessible by the user, as seen in *Figure 8*: Touch Sensor. When pressed, it would stop the robot and drive it in the reverse direction for a short distance to start the chain reaction and knock down the dominoes as described in *Table 1* in section *5.11 Sub-division of tasks*.

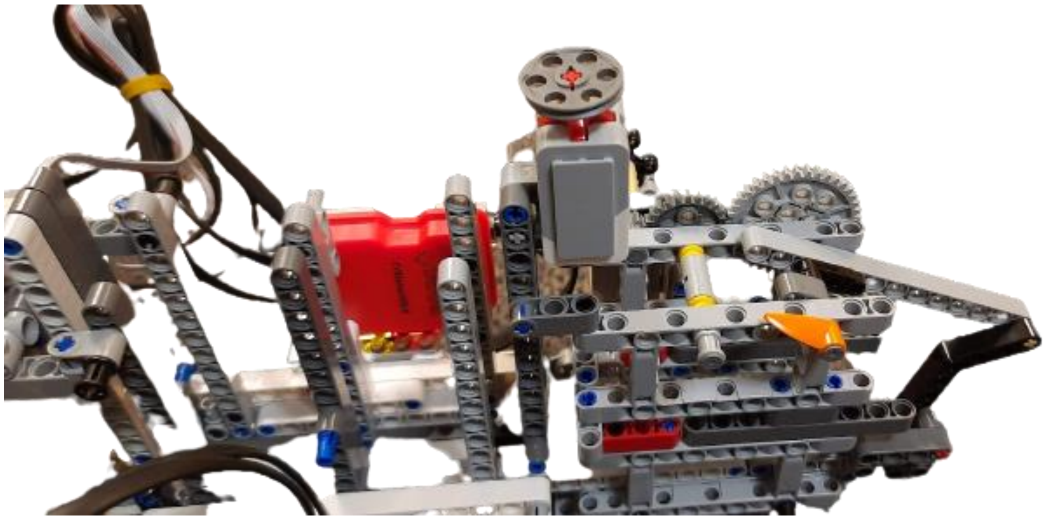

Figure 8: Touch Sensor

4.34 Gyro Sensor

The gyro sensor was originally mounted on one side of the robot opposite to the placement of the medium door motor. However, this caused the robot to leave inconsistent spacings between dominoes when turning, as the sensor would change angles at different ratios depending on turn direction. To solve this, the gyro sensor was remounted on top of the Brick, centred on the robot, as seen in *Figure 9*: Gyro Sensor. This resulted in a more accurate reading. The gyro sensor was used to regulate the rate of rotation of the robot.

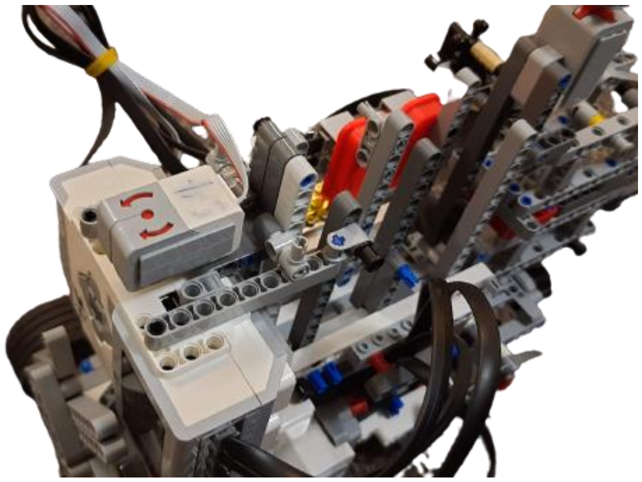

Figure 9: Gyro Sensor

4.4 Overall Assembly

The robot systems were designed to work in unison to successfully dispense dominoes. The drive wheels were mounted to be in line with the domino dispensing chamber to allow for accurate placement on the line. The colour sensors were also attached close to the drive wheel and release chute so this section of the robot would remain over the drawn line. The hopper and dispensing arm were mounted close to the front and center of the robot which pushed dominoes to the back dispensing area. Each sensor was mounted in their respective locations as mentioned in sections *4.31* Color Sensors, *4.32* Ultrasonic Sensor, *4.33* Touch Sensor, *4.34* Gyro Sensor, and as can be seen in *Figure 10*: Full Robot. The EV3 Brick was also mounted above the dispensing area above the drive wheels to help with traction and allowed easy access to the display and buttons.

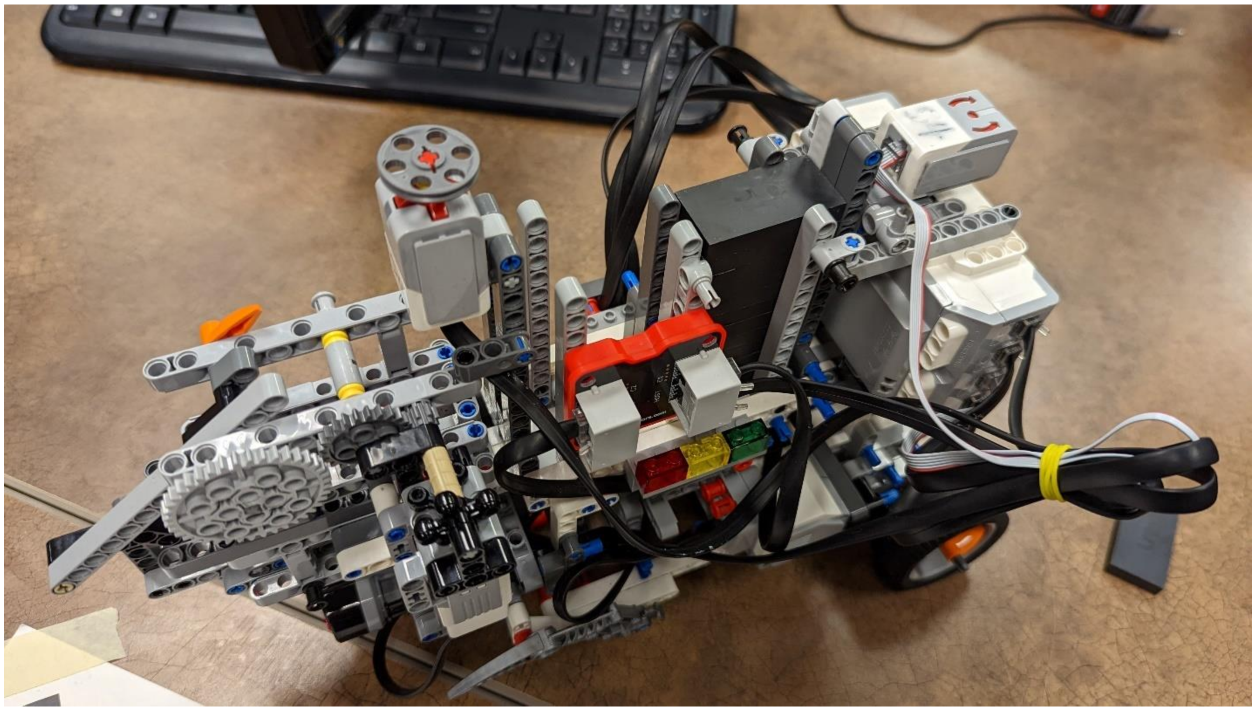

Figure 10: Full Robot

5.0 Software Design and Implementation

5.1 Overall Software Design

5.11 Sub-division of tasks

The main code's function was simplified by the sub-division of tasks into different functions, which were called in the main function as laid out in *Figure 11*: Main Flowchart. This sub-division made the delegation of tasks easier and improved the readability of the code. It also simplified the writing and testing of the code, as any errors could easily be isolated to the specific function causing those errors. This subdivision included 4 general categories, each with several functions: high level functions, calculation functions, one-time functions, and movement functions. These functions are laid out in *Table 1*.

Table 1: List of Functions

Function Name	Return Type	Parameters	Function Description	Programmer
High Level Functions				
followLine()	void	&drop_index, &domino_count	Follows a line on the ground while dropping dominoes (see *Figure 13*: Line Follow Flowchart)	Sean
followPathFromFile()	void	&drop_index, &domino_count	Follows a set of instructions predetermined by the user while dropping dominoes (see *Figure 14*: File Follow Flowchart)	Andor
getInstrFromFile()	int	&all_instr	Loads a set of instructions from a text file into an array of instruction objects, allowing the robot to then follow those instructions with the followPathFromFile() function	Andor
dropDomino()	void	&drop_index, &domino_count	Dispenses a domino from the hopper into the release chute, opening and closing the door as needed (see *Figure 12*: dropDomino() Flowchart)	Henrique

somethingInTheWay()	void	motor_power	Called when an object is detected in front of the robot. Halts all operation until object is removed from in front of the robot, then sets motor speed to continue program operation	Josh
somethingInTheWay()	void	left_mot_pow, right_mot_pow	Does the same as above function, except sets left and right motor speeds separately, allowing this function to be called when turning	Josh
Calculation Functions				
distToDeg()	int	dist	Converts driving distance to a degree value corresponding to motor encoder clicks	Andor
degToDist()	float	deg	Converts motor encoder clicks to distance driven	Andor
average()	float	value1, value2	Returns average value of two separate integers	Sean
One-Time Functions				
configureAllSensors()	void	mode	Configures sensors based on the sensors needed for selected mode (this was done to conserve battery and simplify bug fixing, as using the multiplexer often created issues)	Andor
selectMode()	bool		Allows user to select desired mode using face buttons on the Brick	Andor
endProgram()	void		Runs as the end procedure. Waits for the user to press the touch sensor. If the sensor is pressed, the robot knocks down the domino path it created, if not it exits the program.	Andor

Movement Functions				
setDriveTrainSpeed()	void	speed	Sets the speed of drive motors	Andor
driveDist()	void	dist, mot_pow	Drives specified distance without dropping dominoes	Andor
driveWhileDropping()	void	dist, mot_pow, drop_index, domino_count, dist_since_last_dom	Drives specified distance while dropping dominoes periodically, based on a set distance between dominoes	Andor
turnInPlace()	void	angle, mot_pow	Turns robot in place without dispensing dominoes. Important for followPathFromFile() function	Andor
turnWhileDropping()	void	angle, speed, drop_index, domino_count, dist_since_last_dom	Turns robot through a specific radius while dropping dominoes periodically, based on a set distance between dominoes	Andor
stopAndKnock()	void		Stops robot and drives backwards, knocking down the domino chain	Josh
openDoor()	void		Opens dispenser door	Henrique
closeDoor()	void		Closes dispenser door	Henrique

5.12 Task List

The robot was required to accomplish these specific tasks:

- Follow line
- Read and follow instructions from a file
- Drop dominoes with appropriate spacing
- Respond accordingly to button press
- Stop when an object is seen ahead

Note: as these are very general tasks, they did not change throughout the project.

5.13 Data Storage

Data is stored in three main general categories. Firstly, many constants are used to make adjusting specific parameters in the code very straightforward. Secondly, there are many temporary variables created throughout the code to store important values, such as the number of dominoes remaining in the hopper, or the position of the dispenser arm. Lastly, for the follow path from file mode, a structure was created to simplify the storage of instructions. Each instruction object contains a Boolean representing whether the instruction is a turn or not, and a value. This value represents an angle if the instruction is a turn, or a distance if it is not. This structure is used in conjunction with an array, storing all the instructions for the robot to execute.

5.2 Decisions and Trade-Offs

The codebase was designed with modularity at the forefront. Many functions were used to isolate every separate function of the robot and simplify both the coding process and the readability of the code. While this modular design significantly lengthened the code and led to some integration bugs, the advantages it offered ultimately made the entire coding process much easier.

Aside from this, our program allowed the user to choose between two separate modes, both of which were divided into functions. This added some complexity to the program, as each mode was essentially its own separate program, however it allowed for increased flexibility in how the entire project is used. These modes also had some overlapping functionality, which meant some functions were utilised in both modes.

Another decision that was made was the decision to implement a second colour sensor, bringing the total number of sensors up to five. This required the use of a sensor multiplexer, which increased program complexity significantly and led to many obstructive software errors. Despite this, the use of a second colour sensor significantly increased the accuracy of the robot, as it no longer had to rely only on a gyro sensor (which is significantly less accurate than two colour sensors) to accomplish its line following function.

Structures were also implemented into the program, specifically to aid in the storage of instructions being imported from a text file. This increased the amount of memory each instruction consumed, decreasing the maximum instruction limit. While this was not ideal in terms of memory efficiency, it increased the readability of the code, and the instruction limit would never have been reached, as the robot would have run out of dominoes far before the instruction limit was reached.

5.3 Testing

Functions were tested as laid out in *Table 2*: Function Testing.

Table 2: Function Testing

Function Being Tested	**Reason For Testing**	**Expected Behaviour**	**How Correct Behaviour was Insured**
dropDomino()	To ensure dominoes are dispensed without falling or jamming,	Domino is dispensed, robot waits for domino to	If dominoes were placed consistently and accurately, without

	allowing a proper line of dominoes to be placed accurately and without falling	stabilise, and door is opened	jamming or falling, the function was working correctly
driveWhileDropping()	To ensure that the robot can dispense dominoes with correct, consistent spacing while driving the correct distance	Robot drives specific distance while dropping dominoes at set distance intervals	If dominoes were placed with correct and consistent spacing and robot drives the distance specified, the function was working correctly
turnWhileDropping()	To ensure robot can consistently turn a set radius while dispensing dominoes at a rate that allows dominoes to fall in a chain when knocked over	Robot drives through a specific radius, dropping dominoes at a set interval until it has reached its target angle	If the robot drove through the entire angle, while dropping dominoes at specific, consistent intervals, the function was working correctly
somethingInTheWay()	To ensure robot does not knock over dominoes unintentionally, or run into obstacles during operation	Robot stops anytime an object is detected in front of it, asks user to move object while playing a sound, and continues normal operation when object is no longer detected	If the robot stopped when an object was placed in front of it and continued when the object was removed, the function was working correctly
stopAndKnock()	To ensure robot only knocks dominoes over when it is supposed to and does it consistently when it needs to	Robot stops and backs up until the most recently placed domino is knocked over. It then stops and ends the program.	If dominoes were consistently knocked down whenever the touch sensor was pressed, the function was working correctly
followPathFromFile()	To ensure path specified user is correctly translate to a domino pattern	Robot receives instructions from user through a text file and follows those instructions	If entire path was replicated, or part of the path was replicated (if robot runs out of dominoes before path is complete) in a domino path, the function was working correctly
followLine()	To ensure robot can follow and place dominoes along a given path	Robot places domino, drives following the drawn line to the right	If the line was followed and if the dominoes fell correctly when knocked, the

		spacing, places another domino and repeats	function was working correctly
main()	To ensure all functions are working together properly and all tasks are met	Robot completes all tasks outlined above without knocking dominoes down unintentionally	If the robot followed the path specified by the user and successfully reached a specified end condition, the program was working correctly

5.4 Significant Problems

Significant problems were encountered in two separate instances. Firstly, the RobotC code was initially set up to receive a list of coordinates from a file and then calculate the specific angle and distance instructions locally. This however proved to be very challenging, as the application of linear algebra and other complex concepts was difficult for two reasons. First, writing in a way that was universal in that the same calculations could be used for every set of coordinates, and second, hard to verify as the natural imprecision of the EV3 platform made results difficult to verify. As such the path generation code (written in Python) was modified to output instructions for the robot to follow instead of coordinates. This is because many of the calculations required to find instructions (mainly the angles between lines and the turning direction of the robot) were already calculated by the path generation code. The RobotC code was then also greatly simplified, as it no longer had to do complex calculations, only having to read in instructions and then follow them.

Another issue encountered was the use of the EV3 multiplexer and the reading of values from it. The multiplexer was connected to colour sensors that allowed the robot to follow the line and had to constantly be read from to allow for a smooth following of the line. However, reading from the sensors constantly caused the program to fail. This issue was solved through discussion with teaching assistants and another group which revealed that an individual sensor in the multiplexer could only be read from every five milliseconds. The code was modified to accommodate this timing, allowing the robot to read values from the colour sensor through the multiplexer.

All the code that this section details can be seen in *Appendix A*: Source Code.

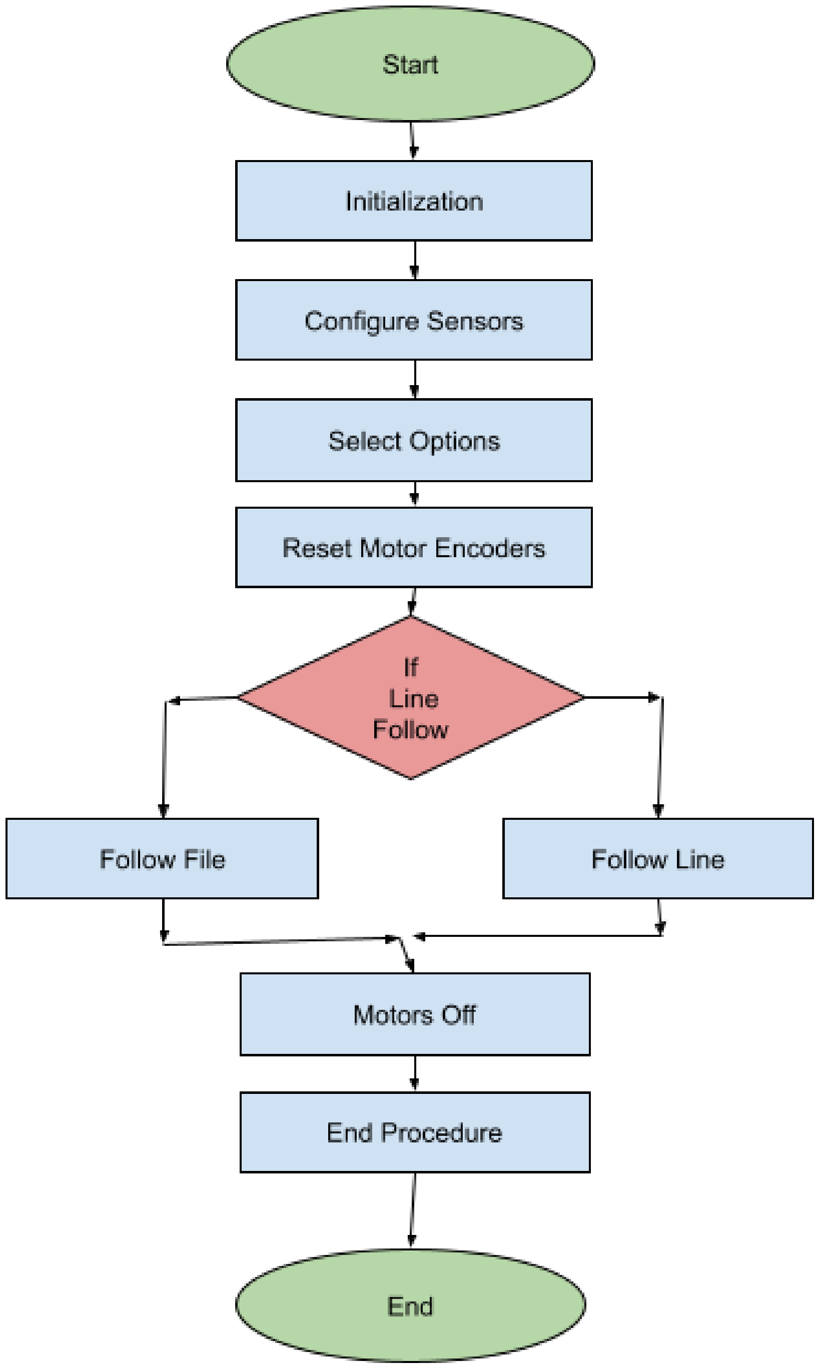

Figure 11: Main Flowchart

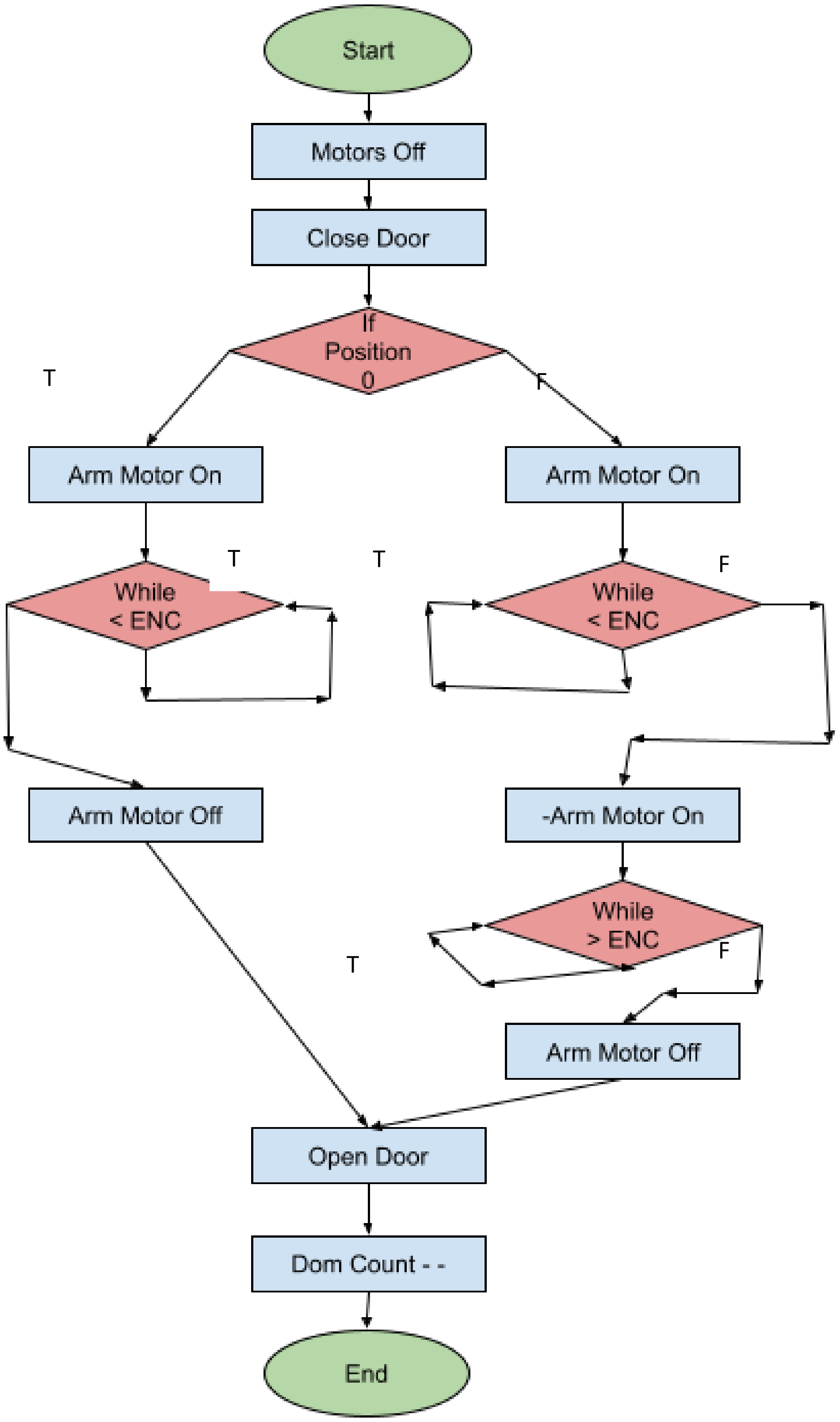

Figure 12: dropDomino() Flowchart

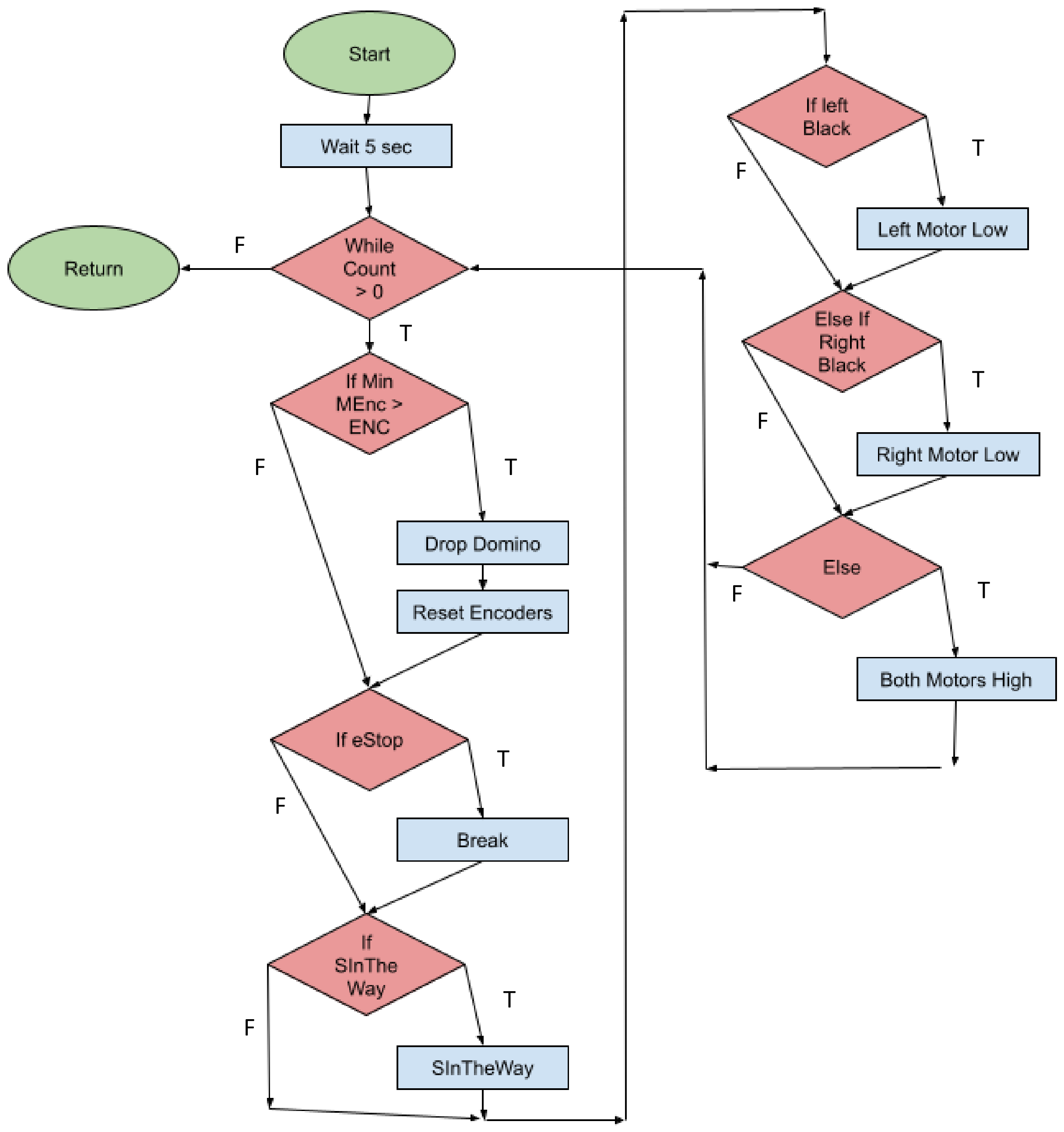

Figure 13: Line Follow Flowchart

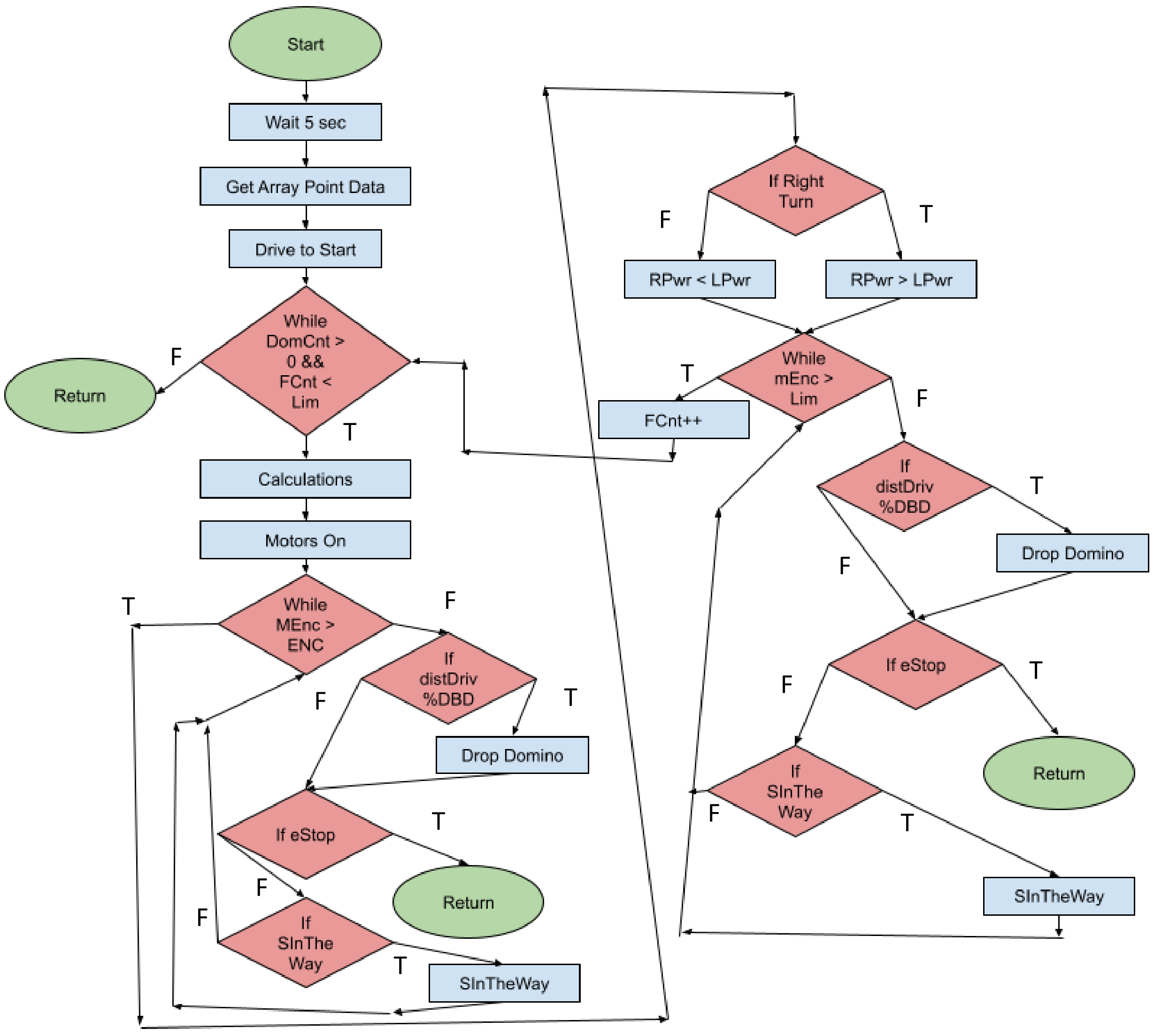

Figure 14: File Follow Flowchart

6.0 Verification

6.1 How Criteria Was Met

The criteria for the robot were to place the dominoes in a well spaced and consistent way, as well as following the designated path and responding to the other outside inputs, such as the touch sensor and the ultrasonic sensor.

To acknowledge that the criteria were met, the following test was done:

1. Initialise the program and choose the line follow mode
2. Let the robot start to follow the line and position an object in front of it and seeing if it responds accordingly to an obstacle
3. Let the robot run out of dominoes and press the touch sensor
4. Observe if the robot positioned the dominoes on the line and whether it responds correctly to the touch sensor input
5. Observe if all the dominoes were knocked down
6. Repeat the test but on file path mode

If all the answers were affirmative, the criteria were met.

6.2 How Constraints Were Decided

To decide on the constraints, the limitations on the mechanical design and on the software were considered. The constraints decided were:

- A domino limit of 30 dominoes
    - This limit was decided as it did not require a stronger motor to overcome the friction caused by more dominoes and allowed for smoother movement.
- A limit of 5 minutes to finish the path
    - This constraint was set because 5 minutes was the maximum amount of time allowed on demonstration day.

7.0 Project Plan

7.1 Delegation

7.11 Software division

The software was divided into two programs, simultaneously being worked on by group members using GitHub. The main RobotC program was split up as follows; Sean was responsible for the main line-follow function; Andor was responsible for the main file-follow function; Henrique was responsible for the dropDomino() function as well as its sub-functions, openDoor() and closeDoor(); Josh was responsible for the somethingInTheWay() function and the stopAndKnock() function. There were also a number of smaller, mostly trivial functions that were written as needed which were not assigned to any particular group member. It should be noted that the majority of these smaller functions, as well as the overall code outline, was completed by Andor. In file-follow mode, this main program received input from a second program written in Python, also by Andor, that created instructions for the robot to follow from a user-defined path.

7.12 Mechanical Delegation

The division of the mechanical work was much less formal than that of the software. The robot itself was mostly worked on simultaneously by all of the group members, making suggestions and putting it together piece by piece. However, some elements were mainly constructed by single group members: the dispenser arm mechanism and much of the outer chassis were designed by Sean, the domino hopper was designed by Josh, and the door mechanism was designed by Henrique.

7.13 Division of Other Tasks

Other tasks such as the acquisition of materials, laying out paths for the robot to follow, creating presentation slides, and testing were all also done by the group as a whole without specific delegation.

7.2 Project Plan Revisions

Overall, the project plan did not change significantly over the course of the project. The only major plan revision was the moving back of the target date for having the codebase completed, from two days before the presentation, to seven days before the presentation, in order to allocate more time for testing and debugging.

7.3 Differences from Project Plan in Implementation

There were few differences between the project plan and what was actually done, although most of them were relatively minor. The code outline and codebase were completed sooner than planned, which allowed for more time dedicated to testing and debugging. Also, some mechanical issues were noticed and adjusted after our target deadline of having the assembly done such as the dispenser described in section *4.22* Dispenser Arm.

8.0 Conclusions

At the outset of this project, the group set out to apply our knowledge and our skills to automate the process of creating a domino chain reaction. This process was to be completed by creating a mechanism that would push dominoes out of a hopper and down a ramp to stand them up, all while moving along a path that was either drawn out on the ground in front of it, or that was passed to the robot through a file. In the completion of the project, the robot successfully accomplished all of its stated goals.

The main criteria for success was for the dominoes to be placed over the chosen path and for those dominoes to be evenly spaced. This was accomplished by using either the colour sensors to follow the line or the Python program to follow the file path, and by taking the average of the two drivetrain motor encoders to space the dominoes. It is clear both by observation, as seen in *Figure 15*: Dominoes Set Up, and by the fact that the dominoes would all successfully topple when desired, that this criterion was met.

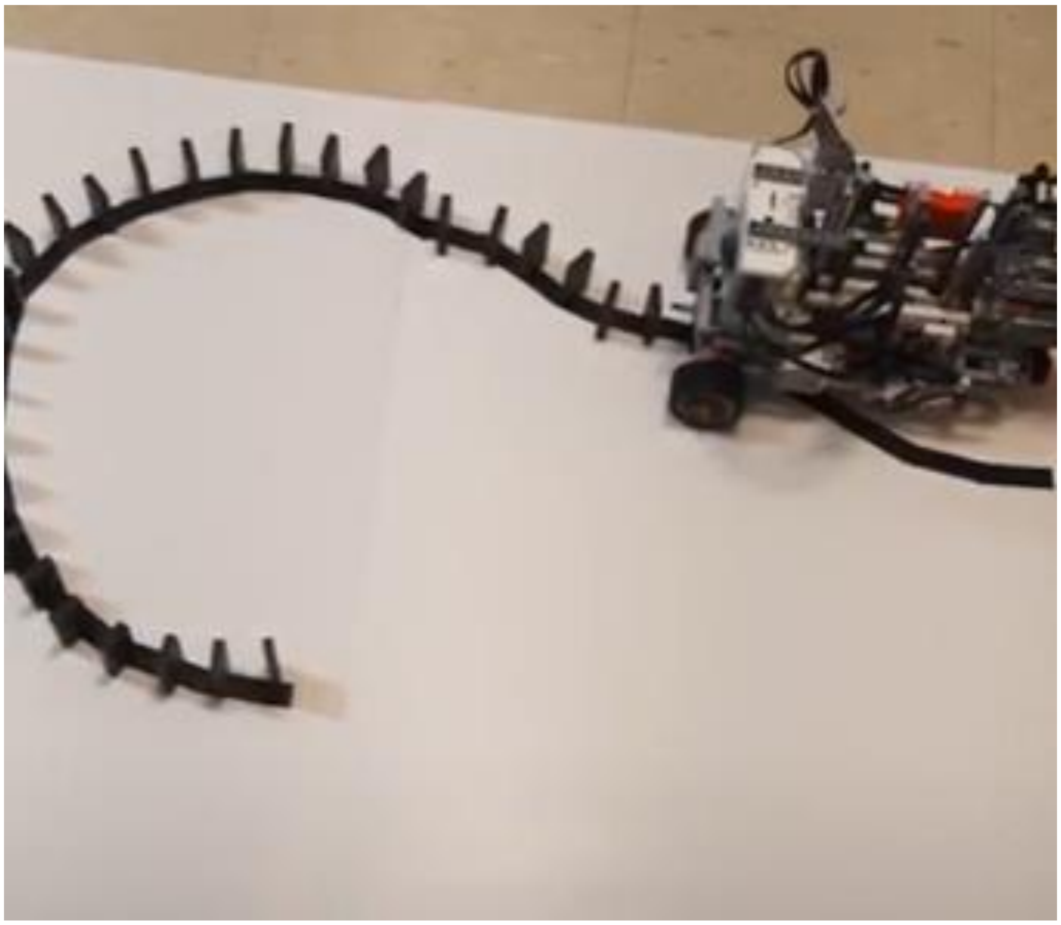

Figure 15: Dominoes Set Up On Line

There were also two constraints on the project: firstly that the robot should be capable of placing at least thirty dominoes, and secondly that it should be able to accomplish this faster than a human. During the final demonstration, the robot was able to hold exactly thirty dominoes, all of which were placed in the allotted five-minute timeframe. This clearly demonstrates that these constraints were met.

9.0 Recommendations

9.1 Mechanical Changes

If this project was to continue in the future, a mechanical redesign would include space for more dominoes, as well as placing the Brick in a different alignment, making it easier to read from the display.

9.2 Software Changes

Alongside the mechanical redesign, software changes would need to be made to accommodate the new mechanical design. Another change that would be appreciated is to modify the Python code to automatically scale the path, so that the path would be completed within the domino limit.

References

[1] ScienceWorksMuseum, "ScienceWorksMuseum," 20 April 2021. [Online]. Available: https://scienceworksmuseum.org/dominoes-endless-possibilities/. [Accessed 18 November 2022].

[2] Z. N. Alekseevich, "zhurov-nikita-physic.weebly," [Online]. Available: https://zhurov-nikita-physic.weebly.com/about.html. [Accessed 18 November 2022].

[3] M. Chauhan, "With Statements in Python," Geeks for Geeks, 18 November 2022. [Online]. Available: https://www.geeksforgeeks.org/with-statement-in-python/.

[4] "Python Write Text File," pythontutorial.net, [Online]. Available: https://www.pythontutorial.net/python-basics/python-write-text-file/.

[5] "Pygame Documentation," pygame, [Online]. Available: http://www.pygame.org/docs/ref/draw.html#pygame.draw.line.

[6] "Python List extend() Method," w3schools, [Online]. Available: https://www.w3schools.com/python/ref_list_extend.asp.

[7] J. Dempsey, "Pygame Drawing a Rectangle," stackoverflow, 24 October 2019. [Online]. Available: https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle.

[8] "How Can I Check If Two Segments Intersect," stackoverflow, 1 November 2022. [Online]. Available: https://stackoverflow.com/questions/3838329/how-can-i-check-if-two-segments-intersect.

[9] "How to check if two given line segments intersect?," geeksforgeeks, 13 July 2022. [Online]. Available: https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/.

[10] A. Ramakrishnan, "Calculating angles between line segments (Python) with math.atan2," stackoverflow, 8 February 2022. [Online]. Available: https://stackoverflow.com/questions/28260962/calculating-angles-between-line-segments-python-with-math-atan2.

[11] Servaes, "calculate the turning radius turning circle of a two wheeled car," stackexchange, 18 November 2021. [Online]. Available: https://math.stackexchange.com/questions/4310012/calculate-the-turning-radius-turning-circle-of-a-two-wheeled-car.

[12] root-11, "How do you display code snippets in MS Word preserving format and syntax highlighting?," stackoverflow, 29 May 2013. [Online]. Available: https://stackoverflow.com/questions/387453/how-do-you-display-code-snippets-in-ms-word-preserving-format-and-syntax-highlig.

[13] X. Soldaat, *mindsensors-ev3smux.h (Version 0.1),* xander_at_botbench.com, 2014.

Appendix A: Source Code

```
/*
Waterloo Engineering Expeller of Dominoes Main Program

Sean Aitken, Henrique Engelke, Josh Morcombe, and Andor Siegers

v1.7

Assumptions:
- more than 3 instructions will be in instruction file
- no more than 100 instructions will be given to the robot
- robot is fully loaded at program start, with exactly 30 dominoes
  in the hopper
- door is closed, dispenser arm is all the way back at program start
- if user is selecting line follow mode, it must be placed on a line of
adequate length
  with white on either side
- if user is selecting file follow mode, a file of the correct format must be
  loaded on the robot

Motor Ports:
A - left drive wheel
B - dispenser motor
C - gate motor
D - right drive wheel

Sensor Ports:
1 - MUX
2 - gyro
3 - touch
4 - ultrasonic
*/

#include "PC_FileIO.c"
#include "mindsensors-ev3smux.h"
#include "UW_sensorMux.c"

typedef struct
{
    bool is_ang;
    int val;

} Instr;

// one-time functions
void configureAllSensors(bool mode);
bool selectMode();
void endProgram();

// high level functions
void followLine(bool &drop_index, int &domino_count); // Sean
void followPathFromFile(bool &drop_index, int &domino_count); // Andor
int getInstrFromFile(Instr* all_instr);
void dropDomino(bool &drop_index, int &domino_count); // Henrique
void somethingInTheWay(int motor_power); // stops and informs the user to
move the object in the way
void somethingInTheWay (int left_mot_pow, int right_mot_pow);

// calculation functions
int distToDeg(float dist);
float degToDist(int deg);
float average(int value1, int value2);


// movement functions
void setDriveTrainSpeed(int speed);
void driveDist(float dist,int mot_pow);
void driveWhileDropping(float dist, int mot_pow, bool &drop_index, int
&domino_count, float &dist_since_last_dom); // Andor
void turnInPlace(int angle, int mot_pow);
void turnWhileDropping(int angle, int speed, bool &drop_index, int
&domino_count, float &dist_since_last_dom); // Andor
void stopAndKnock(); // Josh
void openDoor();
void closeDoor();

// constants
const float WHEEL_RAD = 2.75; // in cm
const int DOMINOS_AT_MAX_LOAD = 30;
const int MAX_INSTR = 100;
const float PIXELS_PER_CM = 5.0;
const float DIST_BETWEEN_DOMINOS = 3.75; // in cm
const float DIST_BET_DOM_TURNING = 5.5; // in cm
const int DRIVE_SPEED = 20; // for path from file mode
const int DIST_IN_FRONT_LIM = 20; // in cm
const float TURN_RAD = 20; // in cm - needs to be more than 6.75cm
const int TIME_TO_PRESS = 10; // in seconds
const int DOOR_ANG = 90; // degrees
const int DOOR_SPEED = 50;
const int DROP_WAIT = 500; // in milliseconds
const int MUX_WAIT = 10;
const int DISPENSER_SPEED = -30;
const int DISPENSER_POS0 = 80;
const int DISPENSER_POS1 = -370;
const int DISPENSER_POS2 = -530;
const int KNOCK_SPEED = -15;

// port assignments
const int TOUCH_PORT = S2;
const int GYRO_PORT = S3;
const int MULTIPLEXER_PORT = S1;
const int ULTRASONIC_PORT = S4;

const int RIGHT_MOT_PORT = motorD;
const int LEFT_MOT_PORT = motorA;
const int DOOR_MOT_PORT = motorB;
const int DISPENSER_MOT_PORT = motorC;

task main()
{
    // initialization for domino dropping
    nMotorEncoder(DISPENSER_MOT_PORT) = 0;
    nMotorEncoder(DOOR_MOT_PORT) = 0;
    bool drop_index = false; // false for back position, true for middle
position
    int domino_count = DOMINOS_AT_MAX_LOAD;

    if(selectMode())// false for line follow, true for file path
    {
        followPathFromFile(drop_index, domino_count);
    }
    else
    {
        followLine(drop_index, domino_count);
    }
}

//  ********************************** one-time functions
*************************
void configureAllSensors(bool mode)
{
    SensorType[TOUCH_PORT] = sensorEV3_Touch;
    SensorType[GYRO_PORT] = sensorEV3_Gyro;
    wait1Msec(50);
    SensorType[ULTRASONIC_PORT] = sensorEV3_Ultrasonic;
    wait1Msec(50);
    SensorMode[GYRO_PORT] = modeEV3Gyro_Calibration;
    wait1Msec(50);
    SensorMode[GYRO_PORT] = modeEV3Gyro_RateAndAngle;
    wait1Msec(50);

    // if line follow mode is selected, configure sensors required for
    // this mode
    if(!mode)
    {
        SensorType[MULTIPLEXER_PORT] = sensorEV3_GenericI2C;
        wait1Msec(100);

        if (!initSensorMux(msensor_S1_1, colorMeasureColor))
        {
            displayString(2,"Failed to configure colour1");
            return;
        }
        wait1Msec(50);
        if (!initSensorMux(msensor_S1_2, colorMeasureColor))
        {
            displayString(4,"Failed to configure colour2");
            return;
        }
        wait1Msec(50);
    }
}

bool selectMode()
{
    displayTextLine(5, "Choose Mode");
    displayTextLine(7, "Left - Follow Line");
    displayTextLine(9, "Right - Follow Path from File");
    while(!getButtonPress(buttonLeft) && !getButtonPress(buttonRight))
    {}

    // returns true if buttonRight is pressed (path from file mode)
    // returns false if buttonLeft is pressed (line follow mode)
    bool mode = getButtonPress(buttonRight);
    configureAllSensors(mode);
    wait1Msec(700);
    return mode;
}

void endProgram()
{
    setDriveTrainSpeed(0);
    time1[T1] = 0;
    // wait for user to press touch sensor
    while(time1[T1] < TIME_TO_PRESS*1000)
    {
        if(SensorValue[TOUCH_PORT])
            stopAndKnock();
    }
    stopAllTasks();
}

// ********************************** high level functions
*************************
void followLine(bool &drop_index, int &domino_count) // Sean
{
    time1[T2] = 0;
    int index = 0;
    int index2 = 0;
    int sensor1 = 0;
    int sensor2 = 0;
    int domino_encoder_spacing = distToDeg(DIST_BETWEEN_DOMINOS);

    openDoor();

    while((domino_count>0)&&(SensorValue(TOUCH_PORT) == 0))
    {

        if((SensorValue[ULTRASONIC_PORT]) < (DIST_IN_FRONT_LIM))
        {
            somethingInTheWay(0);
        }


if((average(nMotorEncoder[RIGHT_MOT_PORT],nMotorEncoder[LEFT_MOT_PORT])) >
domino_encoder_spacing)
        {
            dropDomino(drop_index, domino_count);
            nMotorEncoder[RIGHT_MOT_PORT] = nMotorEncoder[LEFT_MOT_PORT] = 0;
        }

        motor[LEFT_MOT_PORT] = motor[RIGHT_MOT_PORT] = -10;

        if(time1[T2] > index)
        {
            sensor1 = readMuxSensor(msensor_S1_1);
            index = time1[T2] + MUX_WAIT;

            if(sensor1 == (int) colorBlack)
            {
                motor[RIGHT_MOT_PORT] = 0;
            }
        }

        if(time1[T2] > index2)
        {
            sensor2 = readMuxSensor(msensor_S1_2);
            index2 = time1[T2] + MUX_WAIT + 5;

            if(sensor2 == (int) colorBlack)
            {
                motor[LEFT_MOT_PORT] = 0;
            }
        }
    }
    if(SensorValue(TOUCH_PORT))
    {
        stopAndKnock();
    }
    endProgram();
}

void followPathFromFile(bool &drop_index, int &domino_count) // Andor
{
    Instr all_instr[MAX_INSTR];
    float dist_since_last_dom = 0;

    int num_instr = getInstrFromFile(all_instr);

    int num_turns = 0;
    int instr_index = 0;

    // drive to starting position
    while(num_turns < 2)
    {
        if(all_instr[instr_index].is_ang)
        {
            num_turns++;
            turnInPlace(all_instr[instr_index].val, 20);
        }
        else
        {
            driveDist(all_instr[instr_index].val/PIXELS_PER_CM, 50);
        }
        instr_index++;
    }

    while(instr_index < num_instr && domino_count > 0)
    {
        // loop through all instructions
        if(all_instr[instr_index].is_ang)
        {
            // turn
            turnWhileDropping(all_instr[instr_index].val, DRIVE_SPEED,
drop_index, domino_count, dist_since_last_dom);
        }
        else
        {
            // drive length
            driveWhileDropping(all_instr[instr_index].val/PIXELS_PER_CM,
DRIVE_SPEED, drop_index, domino_count, dist_since_last_dom);
        }
        instr_index++;
    }
    endProgram();
}

int getInstrFromFile(Instr* all_instr) // Andor
{
    // open file and initialize variables
    TFileHandle fin;
    bool fileOkay = openReadPC(fin,"instr.txt");

    int num_instr = 0;
    readIntPC(fin, num_instr);

    int temp_is_ang_int = 0;
    bool temp_is_ang = false;
    int temp_val = 0;

    for(int read_index = 0; read_index < num_instr; read_index++)
    {
        // read in instruction
        readIntPC(fin, temp_is_ang_int);
        if(temp_is_ang_int == 0)
        {
            temp_is_ang = false;
        }
        else
        {
            temp_is_ang = true;
        }

        readIntPC(fin, temp_val);
        all_instr[read_index].is_ang = temp_is_ang;
        all_instr[read_index].val = temp_val;
    }

    closeFilePC(fin);
    return num_instr;
}

void dropDomino(bool &drop_index, int &domino_count) // Henrique
{
    setDriveTrainSpeed(0);
    closeDoor();
    // moves dispenser arm to next position, depending on current
    // position
    if (!drop_index)
    {
        motor[DISPENSER_MOT_PORT] = DISPENSER_SPEED;
        while (nMotorEncoder(DISPENSER_MOT_PORT) > DISPENSER_POS1)
        {
            // scan for touch press
            if(SensorValue[TOUCH_PORT])
            {
                motor[DISPENSER_MOT_PORT] = 0;
                stopAndKnock();
            }
        }
        motor[DISPENSER_MOT_PORT] = 0;
        drop_index = true;
        wait1Msec(DROP_WAIT);
    }
    else
    {
        motor[DISPENSER_MOT_PORT] = DISPENSER_SPEED;
        while (nMotorEncoder(DISPENSER_MOT_PORT) > DISPENSER_POS2)
        {
            if(SensorValue[TOUCH_PORT])
            {
                motor[DISPENSER_MOT_PORT] = 0;
                stopAndKnock();
            }
        }
        motor[DISPENSER_MOT_PORT]= 0;

        drop_index = false;
        wait1Msec(100);

        // reset arm to initial position
        motor[DISPENSER_MOT_PORT] = -DISPENSER_SPEED;
        while (nMotorEncoder(DISPENSER_MOT_PORT) < DISPENSER_POS0)
        {
            if(SensorValue[TOUCH_PORT])
            {
                motor[DISPENSER_MOT_PORT] = 0;
                stopAndKnock();
            }
        }
        motor[DISPENSER_MOT_PORT] = 0;
    }
    openDoor();
    domino_count--;
}

void somethingInTheWay (int motor_power) // Josh
{
    // Stops motors, displays message and plays a sound. Exits when object is
moved.
    while(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
    {
        setDriveTrainSpeed(0);
        eraseDisplay();
        displayString(5, "Please clear path ahead");
        playSound(soundBeepBeep);
    }
    ev3StopSound();
    setDriveTrainSpeed(motor_power);
}

void somethingInTheWay (int left_mot_pow, int right_mot_pow)
{
    // same as apove, just with 2 motor inputs to accomodate the
    // use of this function in turns
    while(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
    {
        setDriveTrainSpeed(0);
        eraseDisplay();
        displayString(5, "Please clear path ahead");
        playSound(soundBeepBeep);
    }
    ev3StopSound();
    motor[LEFT_MOT_PORT] = left_mot_pow;
    motor[RIGHT_MOT_PORT] = right_mot_pow;
}

// ********************************** calculation functions
*************************
int distToDeg(float dist)
{
    // takes a distance and converts it to motor encoder clicks
    // using wheel radius
    return dist*180/PI/WHEEL_RAD;
}

float degToDist(int deg)
{
    // converts degrees to motor encoder clicks using wheel radius
    return deg*PI*WHEEL_RAD/180;
}

float average(int value1, int value2)
{
    // returns average of two fucntions
    return (abs(value1 + value2)/2.0);
}

// ********************************** movement functions
*************************
void setDriveTrainSpeed(int speed)
{
    // accomodates the backwards mounting of drive motors
    motor[LEFT_MOT_PORT] = motor[RIGHT_MOT_PORT] = -1*speed;
}

void driveDist(float dist, int mot_pow)
{
    // drives specified distance without dropping dominoes
    setDriveTrainSpeed(mot_pow);
    nMotorEncoder[LEFT_MOT_PORT] = 0;
    while(abs(nMotorEncoder[LEFT_MOT_PORT]) < distToDeg(dist))
    {
        // check for break conditions
        if(SensorValue[TOUCH_PORT])
        {
            stopAndKnock();
        }
        else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
        {
            somethingInTheWay(mot_pow);
        }
    }
    setDriveTrainSpeed(0);
}

void driveWhileDropping(float dist, int mot_pow, bool &drop_index, int
&domino_count, float &dist_since_last_dom)
{
    // drives specified distance while dropping dominos at consistent
intervals
    setDriveTrainSpeed(mot_pow);
    nMotorEncoder[LEFT_MOT_PORT] = 0;
    nMotorEncoder[RIGHT_MOT_PORT] = 0;
    while(degToDist(abs(nMotorEncoder(LEFT_MOT_PORT))) < dist && domino_count
> 0)
    {
        // check for break conditions
        if(SensorValue[TOUCH_PORT])
        {
            stopAndKnock();
        }
        else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
        {
            somethingInTheWay(mot_pow);
        }

        // drop domino every DIST_BETWEEN_DOMINOS
        if(degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT))) +
dist_since_last_dom >= DIST_BETWEEN_DOMINOS)
        {
            dist_since_last_dom = 0;
            nMotorEncoder(RIGHT_MOT_PORT) = 0;
            dropDomino(drop_index, domino_count);
            setDriveTrainSpeed(mot_pow);
        }
    }
    // allows for a smooth transition in the domino path between driving
linearly and turning
    dist_since_last_dom = degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT)));
}

void turnInPlace(int angle, int mot_pow)
{
    int initialGyro = getGyroDegrees(GYRO_PORT);
    if(angle < 0)
    {
        // turn left
        motor[LEFT_MOT_PORT] = mot_pow;
        motor[RIGHT_MOT_PORT] = -mot_pow;
        while(getGyroDegrees(GYRO_PORT) > initialGyro+angle)
        {
            // check for break conditions
            if(SensorValue[TOUCH_PORT])
            {
                stopAndKnock();
            }
            else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
            {
                somethingInTheWay(mot_pow, -mot_pow);
            }
        }
    }
    else if(angle > 0)
    {
        // turn right
        motor[LEFT_MOT_PORT] = -mot_pow;
        motor[RIGHT_MOT_PORT] = mot_pow;
        while(getGyroDegrees(GYRO_PORT) < initialGyro+angle)
        {
            // check for break conditions
            if(SensorValue[TOUCH_PORT])
            {
                stopAndKnock();
            }
            else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
            {
                somethingInTheWay(-mot_pow, mot_pow);
            }
        }
}

    setDriveTrainSpeed(0);
}

void turnWhileDropping(int angle, int speed, bool &drop_index, int
&domino_count, float &dist_since_last_dom)
{
    // some concepts taken from:
    // https://math.stackexchange.com/questions/4310012/calculate-the-
turning-radius-turning-circle-of-a-two-wheeled-car

    // turns the robot through a specific radius while dropping dominoes
    float const TURN_RATIO = (TURN_RAD-13.5)/TURN_RAD;
    int initialGyro = getGyroDegrees(GYRO_PORT);
    if(angle > 0)
    {
        // turn Right
        motor[LEFT_MOT_PORT] = -speed;
        motor[RIGHT_MOT_PORT] = -speed*TURN_RATIO;
        nMotorEncoder(LEFT_MOT_PORT) = 0;
        while(getGyroDegrees(GYRO_PORT) < initialGyro+angle && domino_count >
0)
        {
            // check for break conditions
            if(SensorValue[TOUCH_PORT])
            {
                stopAndKnock();
            }
            else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
            {
                somethingInTheWay(-speed, -speed*TURN_RATIO);
            }

            if(degToDist(abs(nMotorEncoder(LEFT_MOT_PORT))) +
dist_since_last_dom >= DIST_BET_DOM_TURNING)
            {
                // drops domino if correct spacing is reached
                dist_since_last_dom = 0;
                nMotorEncoder(LEFT_MOT_PORT) = 0;
                dropDomino(drop_index, domino_count);
                motor[LEFT_MOT_PORT] = -speed;
                motor[RIGHT_MOT_PORT] = -speed*TURN_RATIO;
            }
        }
        dist_since_last_dom = degToDist(abs(nMotorEncoder(LEFT_MOT_PORT)));
    }
    else if(angle < 0)
    {
        // turn left
        motor[LEFT_MOT_PORT] = -speed*TURN_RATIO;
        motor[RIGHT_MOT_PORT] = -speed;
        nMotorEncoder(RIGHT_MOT_PORT) = 0;
        while(getGyroDegrees(GYRO_PORT) > initialGyro+angle && domino_count >
0)
        {
            // check for break conditions
            if(SensorValue[TOUCH_PORT])
            {
                stopAndKnock();
            }
            else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
            {
                somethingInTheWay(-speed*TURN_RATIO, -speed);
            }
            if(degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT))) +
dist_since_last_dom >= DIST_BET_DOM_TURNING)
            {
                // drops domino if correct spacing is reached
                dist_since_last_dom = 0;
                nMotorEncoder(RIGHT_MOT_PORT) = 0;
                dropDomino(drop_index, domino_count);
                motor[LEFT_MOT_PORT] = -speed*TURN_RATIO;
                motor[RIGHT_MOT_PORT] = -speed;
            }
        }
        dist_since_last_dom = degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT)));
    }
}

void stopAndKnock() // Josh
```

```
{
    // moves backwards, knocking over first domino
    nMotorEncoder(LEFT_MOT_PORT) = 0;
    setDriveTrainSpeed(KNOCK_SPEED);
    while(nMotorEncoder(LEFT_MOT_PORT) < distToDeg(DIST_BETWEEN_DOMINOS-0.5))
    {}
    setDriveTrainSpeed(0);
    stopAllTasks();
}

void openDoor() // Henrique
{
    motor[DOOR_MOT_PORT] = DOOR_SPEED;
    while (nMotorEncoder(DOOR_MOT_PORT)<DOOR_ANG)
    {
        // check for break conditions
        if(SensorValue[TOUCH_PORT])
        {
            motor[DOOR_MOT_PORT] = 0;
            stopAndKnock();
        }
    }
    motor[DOOR_MOT_PORT] = 0;
}

void closeDoor() // Henrique
{
    if(!nMotorEncoder(DOOR_MOT_PORT)<5)
    {
        motor[DOOR_MOT_PORT] = -1*DOOR_SPEED;
        while (nMotorEncoder(DOOR_MOT_PORT)>5)
        {
            // check for break conditions
            if(SensorValue[TOUCH_PORT])
            {
                motor[DOOR_MOT_PORT] = 0;
                stopAndKnock();
            }
        }
        motor[DOOR_MOT_PORT] = 0;
    }
}
```

```
# Path calculator for Waterloo Engineering Expeller of Dominoes

# Andor Siegers

# v1.2

import math
import sys
import pygame
from pygame.locals import *

class Point:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return f'({self.x}, {self.y})'

class Instr:
    def __init__(self, if_ang, val):
        self.if_ang = if_ang
        self.val = val

    def __str__(self):
        return f'{self.if_ang}, {self.val}'

# Finds if 2 given line segments intersect or not
# From: https://www.geeksforgeeks.org/check-if-two-given-line-segments-
intersect/

# Given three collinear points p, q, r, the function checks if
# point q lies on line segment 'pr'
def onSegment(p, q, r):
    if ( (q.x <= max(p.x, r.x)) and (q.x >= min(p.x, r.x)) and
         (q.y <= max(p.y, r.y)) and (q.y >= min(p.y, r.y))):
        return True
    return False

def orientation(p, q, r):
    # to find the orientation of an ordered triplet (p,q,r)
    # function returns the following values:
    # 0 : Collinear points
    # 1 : Clockwise points
    # 2 : Counterclockwise

    # See https://www.geeksforgeeks.org/orientation-3-ordered-points/amp/
    # for details of below formula.

    val = (float(q.y - p.y) * (r.x - q.x)) - (float(q.x - p.x) * (r.y - q.y))
    if (val > 0):

        # Clockwise orientation
        return 1
    elif (val < 0):
        # Counterclockwise orientation
        return 2
    else:

        # Collinear orientation
        return 0

# returns true if the line segment 'p1q1' and 'p2q2' intersect
def doIntersect(p1,q1,p2,q2):

    # Find the 4 orientations required for
    # the general and special cases
    o1 = orientation(p1, q1, p2)
    o2 = orientation(p1, q1, q2)
    o3 = orientation(p2, q2, p1)
    o4 = orientation(p2, q2, q1)

    # General case
    if ((o1 != o2) and (o3 != o4)):
        return True

    # Special Cases

    # p1 , q1 and p2 are collinear and p2 lies on segment p1q1
    if ((o1 == 0) and onSegment(p1, p2, q1)):
        return True

    # p1 , q1 and q2 are collinear and q2 lies on segment p1q1
    if ((o2 == 0) and onSegment(p1, q2, q1)):
        return True

    # p2 , q2 and p1 are collinear and p1 lies on segment p2q2
    if ((o3 == 0) and onSegment(p2, p1, q2)):
        return True

    # p2 , q2 and q1 are collinear and q1 lies on segment p2q2
    if ((o4 == 0) and onSegment(p2, q1, q2)):
        return True

    # If none of the cases
    return False

# returns dot product
def dot(vA, vB):
    return vA[0]*vB[0]+vA[1]*vB[1]

# returns line length
def calcLength(p1, p2):
    return math.sqrt((p1.x-p2.x)**2 + (p1.y-p2.y)**2)

# get angle between two vectors
def getAngle(p1,p2,p3,p4):
    # https://stackoverflow.com/questions/28260962/calculating-angles-
between-line-segments-python-with-math-atan2

    # Get nicer vector form
    lineA = ((p1.x,p1.y),(p2.x,p2.y))
    lineB = ((p3.x,p3.y),(p4.x,p4.y))
    vA = [(lineA[0][0]-lineA[1][0]), (lineA[0][1]-lineA[1][1])]
    vB = [(lineB[0][0]-lineB[1][0]), (lineB[0][1]-lineB[1][1])]
    # Get dot prod
    dot_prod = dot(vA, vB)
    # Get magnitudes
    magA = dot(vA, vA)**0.5
    magB = dot(vB, vB)**0.5
    # Get cosine value
    cos_ = dot_prod/magA/magB
    # Get angle in radians and then convert to degrees
    angle = math.acos(dot_prod/magB/magA)
    # Basically doing angle <- angle mod 360
    ang_deg = math.degrees(angle)%360
    return ang_deg

# calculate the center point of a circle tangent to 2 lines forming an angle
def calcCenterPoint(new_point, rad, coords):
    # from:
    # https://stackoverflow.com/questions/51223685/create-circle-tangent-to-
two-lines-with-radius-r-geometry

    p1 = coords[len(coords) - 2]
    p2 = coords[len(coords) - 1]
    p3 = new_point

    le1 = math.sqrt((p2.x-p1.x)**2 + (p2.y-p1.y)**2) # length of A1-B1
segment
    v1x = (p2.x-p1.x) / le1
    v1y = (p2.y-p1.y) / le1

    le2 = math.sqrt((p3.x-p2.x)**2 + (p3.y-p2.y)**2) # length of A1-B1
segment
    v2x = (p3.x-p2.x) / le2
    v2y = (p3.y-p2.y) / le2

    R = rad
    px1 = p1.x - v1y*R
    py1 = p1.y + v1x*R
    px2 = p2.x - v2y*R
    py2 = p2.y + v2x*R

    px1u = p1.x + v1y*R
    py1u = p1.y - v1x*R
    px2u = p2.x + v2y*R
    py2u = p2.y - v2x*R

    den = v1x*v2y - v2x*v1y

    k1 = (v2y*(px2-px1) - v2x*(py2-py1)) / den
    # k2 = (v1y*(px2-px1) - v1x*(py2-py1)) / den

    k1u = (v2y*(px2u-px1u) - v2x*(py2u-py1u)) / den
    # k2u = (v1y*(px2u-px1u) - v1x*(py2u-py1u)) / den

    tx1 = p1.x + k1*v1x
    ty1 = p1.y + k1*v1y
    # tx2 = p2.x + k2*v2x
    # ty2 = p2.y + k2*v2x

    if(onSegment(p1,Point(tx1,ty1),p2)):
        cx =  px1 + k1*v1x
        cy =  py1 + k1*v1y
        left_turn = False
    else:
        cx =  px1u + k1u*v1x
        cy =  py1u + k1u*v1y
        left_turn = True

    # subtracts length taken from the arc from line lengths
    len_to_sub = calcLength(p2, Point(tx1,ty1))

    return Point(cx,cy), left_turn, len_to_sub

def main():
    # pygame specific instructions from:
    # https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle
    pygame.init()

    DISPLAY = pygame.display.set_mode((700,500),0,32)

    WHITE = (255,255,255)
    BLUE = (0,0,255)
    prev_point = Point(0,0)
    prev_len_to_sub = 0
    ang1 = 0
    line_count = -1
    ANGLE_TOLERANCE = 20
    RADIUS_IN_CM = 20
    PIXELS_PER_CM = 5
    RADIUS_IN_PIXELS = RADIUS_IN_CM*PIXELS_PER_CM
    coords = [] # stores coordinates as point values
    instructs = [] # stores instructions for robot

    DISPLAY.fill(WHITE)

    while True:

        for event in pygame.event.get():
            if (event.type == pygame.KEYDOWN and event.key ==
pygame.K_ESCAPE) or event.type == QUIT:
                # before program ends
                file = open('instr.txt', 'w')
                try:
                    # save instructions to file
                    file.write(str(len(instructs)) + "\n")

                    for i in range(len(instructs)):
                        file.write(str((int)(instructs[i].if_ang)) + " " +
str((int)(instructs[i].val)))
                        if i != len(instructs)-1:
                            file.write("\n")

                except:
                    print("Unable to open file")

                file.close()
                pygame.quit()
                sys.exit()

            if event.type == pygame.MOUSEBUTTONDOWN:
                # when mouse is pressed
                x,y = pygame.mouse.get_pos()
                new_point = Point(x,y)
                # check for double click and continue if it is to avoid
instructions with length 0
                if(new_point.x == prev_point.x and new_point.y ==
prev_point.y):
                    continue
                legal_line = True

                # new line
                p1 = Point(prev_point.x, prev_point.y)
                q1 = Point(new_point.x, new_point.y)

                length = calcLength(new_point, prev_point)

                if line_count == -1:
                    # calculate very first angle to turn
                    angle = math.degrees(math.atan2(new_point.y,new_point.x))
                    ang1 = angle

                elif line_count == 0:
                    # calculates second angle to turn
                    angle = 180-getAngle(new_point, prev_point, Point(0,0),
prev_point)

                    # check if angle is negative
                    ang2 = math.degrees(math.atan2(new_point.y,new_point.x))
                    if ang2 < ang1:
                        angle = -angle

                else:
                    # check if new line lintersects with any other line
                    angle = getAngle(new_point, prev_point,
coords[line_count-1], prev_point)

                    # check if angle between old and new line is more than 20
degrees
                    if angle < ANGLE_TOLERANCE:
                        legal_line = False
                    for i in range(line_count-1):
                        # temp line
                        p2 = coords[i]
                        q2 = coords[i+1]
                        if(doIntersect(p1, q1, p2, q2)):
                            legal_line = False

                if legal_line:
                    # if all checks are passed
                    if line_count != -1:
                        # draws line to visualize path
                        pygame.draw.aaline(DISPLAY, BLUE, (prev_point.x,
prev_point.y), (new_point.x, new_point.y))

                        if line_count >= 1:
                            angle = 180-angle
                            # calculates turn direction, while getting data
to draw circle(representing turning arc)
                            centCoord, left_turn, len_to_sub =
calcCenterPoint(new_point, RADIUS_IN_PIXELS, coords)

                            # adjust angle depending on turn direction
                            if left_turn:
                                angle = -angle

                            # subtract len_to_sub from overall length
                            length -= (len_to_sub + prev_len_to_sub)

                            # subtracts length from previous instruction to
accomodate new arc
                            if line_count == 1 and not
instructs[len(instructs) - 1].if_ang:
                                instructs[len(instructs) - 1].val -=
len_to_sub

                            prev_len_to_sub = len_to_sub

                            # draw circle representing robot turning arc
                            rect = Rect(centCoord.x-RADIUS_IN_PIXELS,
centCoord.y-RADIUS_IN_PIXELS, RADIUS_IN_PIXELS*2, RADIUS_IN_PIXELS*2)
                            pygame.draw.arc(DISPLAY,BLUE,rect,0,2*math.pi, 1)

                        # update display
                        pygame.display.flip()
                    # add new coordinate to point list
                    coords.append(new_point)
                    prev_point = new_point
                    # add new instruction to point list
                    instructs.append(Instr(True,angle))

                    if length > 0:
                        instructs.append(Instr(False, length))

                    line_count += 1

        #  update display
        pygame.display.flip()

main()

# Resources:
# http://www.pygame.org/docs/ref/draw.html#pygame.draw.line
# https://www.geeksforgeeks.org/with-statement-in-python/
# https://www.pythontutorial.net/python-basics/python-write-text-file/
# https://www.w3schools.com/python/ref_list_extend.asp
# https://stackoverflow.com/questions/19780411/pygame-drawing-a-rectangle
# https://stackoverflow.com/questions/3838329/how-can-i-check-if-two-
segments-intersect
# https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/
# https://stackoverflow.com/questions/28260962/calculating-angles-between-
line-segments-python-with-math-atan2
```