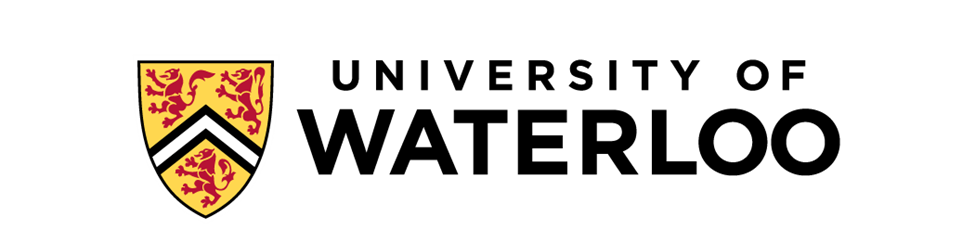
**Group 4 - 20**

**Waterloo** Engineering Expeller **of Dominoes**

****

**Department of Mechanical and Mechatronics Engineering**

**MTE 100 / MTE 121**

**A Report Prepared by:**

Josh Morcombe - 20937588

Andor Siegers - 20990622

Henrique Rodrigues - 21037291

Sean Aitken - 21006546

**Date: Weekday, Day Month, 2022**

**Table Of Contents**

**List Of Figures**

**List Of Tables**

**Acknowledgements**

-domino people

**Summary**

# 1.0 **Introdu**ction

Everyone loves toppling dominoes to create a chain reaction of falling pieces [1], however, it can be really boring and difficult to place them correctly. That is why Group 20 took in their hands the task of automating that process, and create a robot that not only sets the dominoes up straight and consistently spaced, but also creates a determined path to do so.



Figure : Dominoes Being Toppled [2]

**2.0 Scope**

**2.1 Main Functionality**

Our robot completes the chore of setting up dominoes automatically. The path that it will lay the dominoes over is determined either by a line that is drawn on the ground that it will follow, or by a path that is drawn digitally and inputted to the robot by file. The robot then proceeds along the path, intermittently dispensing a domino out behind it from its domino hopper. It will also automatically pause its operation if something is placed in front of it, while alerting the user to the obstacle, proceeding once the obstacle is removed. Finally, the robot will back up and initiate the process of toppling the dominoes when the button on top is pressed. This can happen either in the middle of its operation, or once it has run out of dominoes, at which point it would complete its shutdown. If the user wishes to topple the dominoes manually, this is also possible, as the robot will still shut down automatically after a brief waiting period if the button is never pressed.

**2.2 Inputs**

The robot has a number of inputs that it uses in its regular operation. Firstly, it takes information from the buttons on the EV3 brick in order to select whether it will be operating in file follow mode or line follow mode. In file follow mode, it will receive a file that has been previously generated and perform calculations that will dictate its movements accordingly. In path follow mode, it will receive information from the two color sensors on the bottom and use that to follow a line on the ground. Regardless of the mode, it will also take continuous inputs from the ultrasonic sensor to detect objects, and from the touch sensor to initiate the end procedure.

**2**.3 Interaction with the Environment

The robot has 4 motors that allow it to interact with its environment and complete actions. Firstly, two large motors are connected to the wheels. These allow the robot to move across the ground in two dimensions and thereby complete its paths. There is also one medium motor connected to a pusher arm that pushes one domino at a time from the domino hopper down the ramp and into position. Finally, another medium motor that operates the door that allows the dominoes to escape out the back of the robot at regular intervals.

**2.4** Shutdown Procedure

The general operation of the robot consists of one of two main operational loops, depending on the selected mode. One way to exit this loop is for the robot to run out of dominoes; there is an internal count stored as an integer variable that will decrement each time that a domino is released, and the robot will enter its shutdown procedure once this value reaches zero. Alternatively, the loop can also be exited if the touch sensor is triggered, at which point the robot would automatically topple the first domino and shutdown. Until one of these two things happen, the robot will continue to operate.

The shutdown procedure, triggered only when the robot runs out of dominoes, is as follows: the robot will first stop its operations, that is, all motors will be turned off. The robot will then wait for the touch sensor to be pressed, which would make it topple the first domino. If the touch sensor is not pressed within ten seconds, however, the robot will assume that the dominos are being toppled manually, and will then shut down.

**2.5 Changes to the Scope**

The largest change that we have made to the scope of the project since its inception was the decision to use two color sensors to follow the line instead of one. While we were initially hoping to get away with just one, we realized upon doing a rough outline of the code the it would be very difficult if not impossible to create a reliable path-following program with only one input, so we altered the plan to use two color sensors to make it more realistic and achievable.

We also made a small alteration to the function of the touch sensor; originally, the touch sensor was only going to be an estop button, but we decided it would be cooler if it also made the robot topple the first domino, and it was an easy change, so we decided to go forward with the idea.

**3.0 Constraints and Criteria**

**3.1 Constraints**

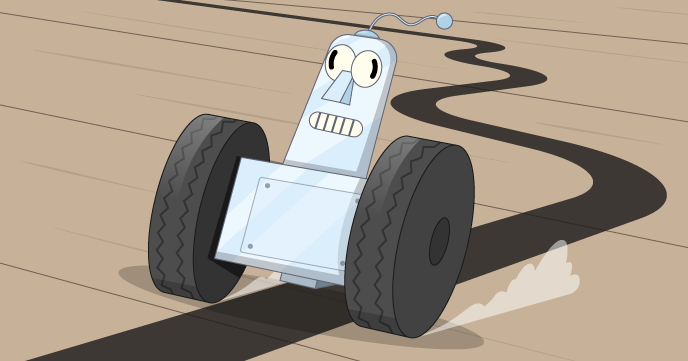
The robot has two main constraints, the amount of dominoes it can carry, and the time of the demo. The robot can carry up to 30 dominoes, and has to place them in less than 5 minutes, because that is the maximum time for the demonstration.

Over the course of the project, those constraints were changed. Initially the maximum number of dominoes to be carried was 60, because that is the amount we bought, however, it proved impossible to carry that many pieces because of the weight it created on top of the other dominoes, increasing friction. The initial time the group planned was 10 minutes, however it was later informed that the demo was only 5 minutes long.

**3.2 Criteria**

The group expects the robot to place the dominoes correctly and evenly spaced from each other and fall all together when one is knocked over. It is also expected to follow a line while setting the dominoes or follow a path from a file uploaded to it.

The criteria were not changed during the development of the project, and remains the same since its inception.



**4.0 Mechanical** Design and Implementation

[Talk about Overall design]

## 4.10 Chassis design

[general chassis]

### 4.11 Hopper

### 4.12 Ramp

## 4.2 Motor Drive Design

### 4.21 Drivetrain

### 4.22 Pusher Arm

### 4.23 Door

## 4.3 Sensor Attachment Design

### 4.31 Color Sensors

### 4.32 Ultrasonic Sensor

### 4.33 Touch Sensor

### 4.34 Gyro Sensor

## 4.4 Overall Assembly

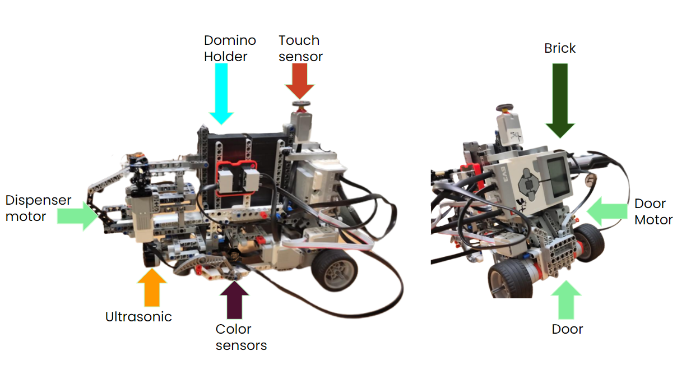
****

Figure : Full Robot Sideview

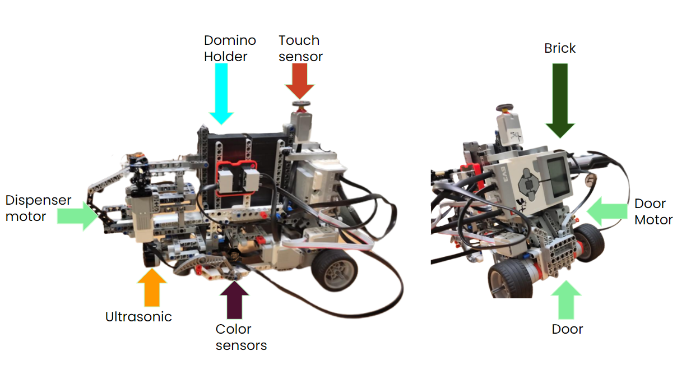
****

Figure : Full Robot Back View

**5.0 Software Design and Implementation**

## 5.1 Overall Software Design

### 5.11 Sub-division of tasks

### 5.12 Task List

### 5.13 Functions

### 5.14 Data Storage

## 5.2 Decisions and Trade-Offs

## 5.3 Testing

## 5.4 Significant Problems

Diagram

Description automatically generated

Figure : Macro Flowchart

Diagram

Description automatically generated

Figure : Drop Domino Flowchart

Diagram

Description automatically generated

Figure : Line Follow Flowchart

Diagram

Description automatically generated

Figure : File Follow Flowchart

**6.0 Verification**

**7.0 Project Plan**

**7.1 Delegation**

### 7.11 Software division

The software program was divided into two programs, being worked by the group member simultaneously using GitHub. Each member designed several functions for the main program. Those functions coordinated with the Python program written by Andor to create the coordinates for the robot to follow.

### 7.12 Division of Mechanical Work

### 7.13 Division of Other Tasks

## 7.2 Revisions

## 7.3 Changes From Project Plan

**8.0 Conclusions**

**9.0 Recommendations**

## 9.1 Mechanical Changes

## 9.2 Software Changes

**References**

1. <https://scienceworksmuseum.org/dominoes-endless-possibilities/>
2. Image introduction: <https://zhurov-nikita-physic.weebly.com/lab-84703-the-domino-effect-lab.html>
3. Image criteria: https://projects.raspberrypi.org/en/projects/rpi-python-line-following/4

**Appendix A: Source Code**

/\*

Domino layer path follower

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

v1.5

Assumptions:

- more than 3 instructions will be in instruction file

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(!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;

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

{

// TODO add break conditions to this function

// DO NOT DROP DOMINOES FOR FIRST INSTRUCTION

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

{

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++)

{

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();

if (!drop\_index)

{

motor[DISPENSER\_MOT\_PORT] = DISPENSER\_SPEED;

while (nMotorEncoder(DISPENSER\_MOT\_PORT) > DISPENSER\_POS1)

{

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);

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--;

// continue line or path follow after

}

void somethingInTheWay (int motor\_power) // Josh

{

// takes UltraSonic sensor port, max distance from an object and motor power.

// Stops motors, displays message and plays a sound. continues when object is moved.

while(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM)

{

setDriveTrainSpeed(0);

eraseDisplay();

displayString(5, "Please clear path ahead");

playSound(soundBeepBeep); // can change later

}

ev3StopSound();

setDriveTrainSpeed(motor\_power);

}

void somethingInTheWay (int left\_mot\_pow, int right\_mot\_pow)

{

// takes UltraSonic sensor port, max distance from an object and motor power.

// Stops motors, displays message and plays a sound. continues when object is moved.

while(SensorValue[ULTRASONIC\_PORT] < DIST\_IN\_FRONT\_LIM)

{

setDriveTrainSpeed(0);

eraseDisplay();

displayString(5, "Please clear path ahead");

playSound(soundBeepBeep); // can change later

}

ev3StopSound();

motor[LEFT\_MOT\_PORT] = left\_mot\_pow;

motor[RIGHT\_MOT\_PORT] = right\_mot\_pow;

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* calculation functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int distToDeg(float dist)

{

return dist\*180/PI/WHEEL\_RAD;

}

float degToDist(int deg)

{

return deg\*PI\*WHEEL\_RAD/180;

}

float average(int value1, int value2)

{

return (abs(value1 + value2)/2.0);

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* movement functions \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void setDriveTrainSpeed(int speed)

{

motor[LEFT\_MOT\_PORT] = motor[RIGHT\_MOT\_PORT] = -1\*speed;

}

void driveDist(float dist, int mot\_pow)

{

// input negative motor power for backwards

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)

{

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);

}

}

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)

{

// https://math.stackexchange.com/questions/4310012/calculate-the-turning-radius-turning-circle-of-a-two-wheeled-car

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)

{

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)

{

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 to knock 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;

}

}