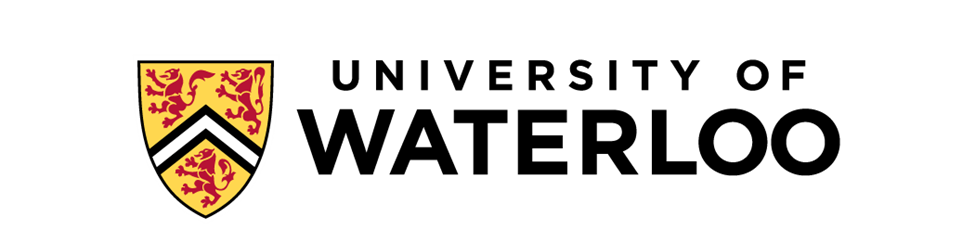
**Group 4 - 20**

**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**

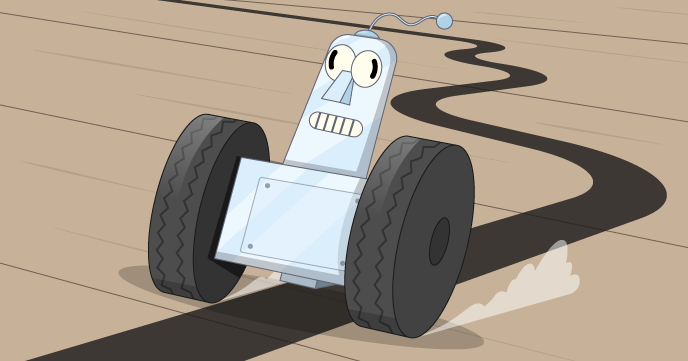
The robot has two main constraints, the amount of dominoes it can carry, and the time of the demo. The robot can carry up to 30 dominoes, and has to place them in less than 5 minutes, because that is the maximum time for the demonstration.

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

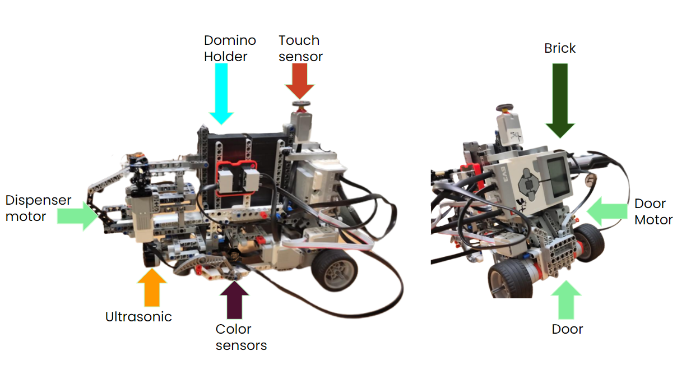
****

Figure : Full Robot Sideview

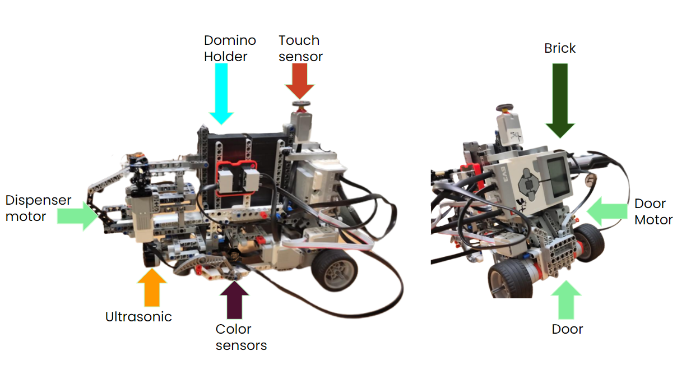
****

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] | 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. |

**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