```
/*
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
```

```
57  int distToDeg(float dist);
58  float degToDist(int deg);
59  float average(int value1, int value2);
60
61
62  // movement functions
63  void setDriveTrainSpeed(int speed);
64  void driveDist(float dist,int mot_pow);
65  void driveWhileDropping(float dist, int mot_pow, bool &drop_index, int &domino_count, float
    &dist_since_last_dom); // Andor
66  void turnInPlace(int angle, int mot_pow);
67  void turnWhileDropping(int angle, int speed, bool &drop_index, int &domino_count, float
    &dist_since_last_dom); // Andor
68  void stopAndKnock(); // Josh
69  void openDoor();
70  void closeDoor();
71
72  // constants
73  const float WHEEL_RAD = 2.75; // in cm
74  const int DOMINOS_AT_MAX_LOAD = 30;
75  const int MAX_INSTR = 100;
76  const float PIXELS_PER_CM = 5.0;
77  const float DIST_BETWEEN_DOMINOS = 3.75; // in cm
78  const float DIST_BET_DOM_TURNING = 5.5; // in cm
79  const int DRIVE_SPEED = 20; // for path from file mode
80  const int DIST_IN_FRONT_LIM = 20; // in cm
81  const float TURN_RAD = 20; // in cm - needs to be more than 6.75cm
82  const int TIME_TO_PRESS = 10; // in seconds
83  const int DOOR_ANG = 90; // degrees
84  const int DOOR_SPEED = 50;
85  const int DROP_WAIT = 500; // in milliseconds
86  const int MUX_WAIT = 10;
87  const int DISPENSER_SPEED = -30;
88  const int DISPENSER_POS0 = 80;
89  const int DISPENSER_POS1 = -370;
90  const int DISPENSER_POS2 = -530;
91  const int KNOCK_SPEED = -15;
92
93  // port assignments
94  const int TOUCH_PORT = S2;
95  const int GYRO_PORT = S3;
96  const int MULTIPLEXER_PORT = S1;
97  const int ULTRASONIC_PORT = S4;
98
99  const int RIGHT_MOT_PORT = motorD;
100 const int LEFT_MOT_PORT = motorA;
101 const int DOOR_MOT_PORT = motorB;
102 const int DISPENSER_MOT_PORT = motorC;
103
104 task main()
105 {
106         // initialization for domino dropping
107         nMotorEncoder(DISPENSER_MOT_PORT) = 0;
108         nMotorEncoder(DOOR_MOT_PORT) = 0;
109         bool drop_index = false; // false for back position, true for middle position
110         int domino_count = DOMINOS_AT_MAX_LOAD;
111
112         if(selectMode())// false for line follow, true for file path
```

```
113         {
114                 followPathFromFile(drop_index, domino_count);
115         }
116         else
117         {
118                 followLine(drop_index, domino_count);
119         }
120 }
121
122 //  ********************************* one-time functions ************************
123 void configureAllSensors(bool mode)
124 {
125         SensorType[TOUCH_PORT] = sensorEV3_Touch;
126         SensorType[GYRO_PORT] = sensorEV3_Gyro;
127         wait1Msec(50);
128         SensorType[ULTRASONIC_PORT] = sensorEV3_Ultrasonic;
129         wait1Msec(50);
130         SensorMode[GYRO_PORT] = modeEV3Gyro_Calibration;
131         wait1Msec(50);
132         SensorMode[GYRO_PORT] = modeEV3Gyro_RateAndAngle;
133         wait1Msec(50);
134
135         // if line follow mode is selected, configure sensors required for
136         // this mode
137         if(!mode)
138         {
139                 SensorType[MULTIPLEXER_PORT] = sensorEV3_GenericI2C;
140                 wait1Msec(100);
141
142                 if (!initSensorMux(msensor_S1_1, colorMeasureColor))
143                 {
144                         displayString(2,"Failed to configure colour1");
145                         return;
146                 }
147                 wait1Msec(50);
148                 if (!initSensorMux(msensor_S1_2, colorMeasureColor))
149                 {
150                         displayString(4,"Failed to configure colour2");
151                         return;
152                 }
153                 wait1Msec(50);
154         }
155 }
156
157 bool selectMode()
158 {
159         displayTextLine(5, "Choose Mode");
160         displayTextLine(7, "Left - Follow Line");
161         displayTextLine(9, "Right - Follow Path from File");
162
163         while(!getButtonPress(buttonLeft) && !getButtonPress(buttonRight))
164         {}
165
166         // returns true if buttonRight is pressed (path from file mode)
167         // returns false if buttonLeft is pressed (line follow mode)
168         bool mode = getButtonPress(buttonRight);
169         configureAllSensors(mode);
```

```
170         wait1Msec(700);
171         return mode;
172 }
173
174 void endProgram()
175 {
176         setDriveTrainSpeed(0);
177         time1[T1] = 0;
178         // wait for user to press touch sensor
179         while(time1[T1] < TIME_TO_PRESS*1000)
180         {
181                 if(SensorValue[TOUCH_PORT])
182                         stopAndKnock();
183         }
184         stopAllTasks();
185 }
186
187 // ******************************** high level functions ************************
188 void followLine(bool &drop_index, int &domino_count) // Sean
189 {
190         time1[T2] = 0;
191         int index = 0;
192         int index2 = 0;
193         int sensor1 = 0;
194         int sensor2 = 0;
195         int domino_encoder_spacing = distToDeg(DIST_BETWEEN_DOMINOS);
196
197         openDoor();
198
199         while((domino_count>0)&&(SensorValue(TOUCH_PORT) == 0))
200         {
201
202                 if((SensorValue[ULTRASONIC_PORT]) < (DIST_IN_FRONT_LIM))
203                 {
204                         somethingInTheWay(0);
205                 }
206
207                 if((average(nMotorEncoder[RIGHT_MOT_PORT],nMotorEncoder[LEFT_MOT_PORT])) >
domino_encoder_spacing)
208                 {
209                         dropDomino(drop_index, domino_count);
210                         nMotorEncoder[RIGHT_MOT_PORT] = nMotorEncoder[LEFT_MOT_PORT] = 0;
211                 }
212
213                 motor[LEFT_MOT_PORT] = motor[RIGHT_MOT_PORT] = -10;
214
215                 if(time1[T2] > index)
216                 {
217                         sensor1 = readMuxSensor(msensor_S1_1);
218                         index = time1[T2] + MUX_WAIT;
219
220                         if(sensor1 == (int) colorBlack)
221                         {
222                                 motor[RIGHT_MOT_PORT] = 0;
223                         }
224                 }
225
226                 if(time1[T2] > index2)
```

```
227                 {
228                         sensor2 = readMuxSensor(msensor_S1_2);
229                         index2 = time1[T2] + MUX_WAIT + 5;
230
231                         if(sensor2 == (int) colorBlack)
232                         {
233                                 motor[LEFT_MOT_PORT] = 0;
234                         }
235                 }
236         }
237         if(SensorValue(TOUCH_PORT))
238         {
239                 stopAndKnock();
240         }
241         endProgram();
242 }
243
244 void followPathFromFile(bool &drop_index, int &domino_count) // Andor
245 {
246         Instr all_instr[MAX_INSTR];
247         float dist_since_last_dom = 0;
248
249         int num_instr = getInstrFromFile(all_instr);
250
251         int num_turns = 0;
252         int instr_index = 0;
253
254         // drive to starting position
255         while(num_turns < 2)
256         {
257                 if(all_instr[instr_index].is_ang)
258                 {
259                         num_turns++;
260                         turnInPlace(all_instr[instr_index].val, 20);
261                 }
262                 else
263                 {
264                         driveDist(all_instr[instr_index].val/PIXELS_PER_CM, 50);
265                 }
266                 instr_index++;
267         }
268
269         while(instr_index < num_instr && domino_count > 0)
270         {
271                 // loop through all instructions
272
273                 if(all_instr[instr_index].is_ang)
274                 {
275                         // turn
276                         turnWhileDropping(all_instr[instr_index].val, DRIVE_SPEED,
    drop_index, domino_count, dist_since_last_dom);
277                 }
278                 else
279                 {
280                         // drive length
281                         driveWhileDropping(all_instr[instr_index].val/PIXELS_PER_CM,
    DRIVE_SPEED, drop_index, domino_count, dist_since_last_dom);
282                 }
```

```
283                 instr_index++;
284         }
285         endProgram();
286 }
287
288 int getInstrFromFile(Instr* all_instr) // Andor
289 {
290         // open file and initialize variables
291         TFileHandle fin;
292         bool fileOkay = openReadPC(fin,"instr.txt");
293
294         int num_instr = 0;
295         readIntPC(fin, num_instr);
296
297         int temp_is_ang_int = 0;
298         bool temp_is_ang = false;
299         int temp_val = 0;
300
301         for(int read_index = 0; read_index < num_instr; read_index++)
302         {
303                 // read in instruction
304                 readIntPC(fin, temp_is_ang_int);
305                 if(temp_is_ang_int == 0)
306                 {
307                         temp_is_ang = false;
308                 }
309                 else
310                 {
311                         temp_is_ang = true;
312                 }
313
314                 readIntPC(fin, temp_val);
315                 all_instr[read_index].is_ang = temp_is_ang;
316                 all_instr[read_index].val = temp_val;
317         }
318
319         closeFilePC(fin);
320         return num_instr;
321 }
322
323 void dropDomino(bool &drop_index, int &domino_count) // Henrique
324 {
325         setDriveTrainSpeed(0);
326         closeDoor();
327
328         // moves dispenser arm to next position, depending on current
329         // position
330         if (!drop_index)
331         {
332                 motor[DISPENSER_MOT_PORT] = DISPENSER_SPEED;
333                 while (nMotorEncoder(DISPENSER_MOT_PORT) > DISPENSER_POS1)
334                 {
335                         // scan for touch press
336                         if(SensorValue[TOUCH_PORT])
337                         {
338                                 motor[DISPENSER_MOT_PORT] = 0;
339                                 stopAndKnock();
```

```
340                         }
341                 }
342                 motor[DISPENSER_MOT_PORT] = 0;
343                 drop_index = true;
344                 wait1Msec(DROP_WAIT);
345         }
346         else
347         {
348                 motor[DISPENSER_MOT_PORT] = DISPENSER_SPEED;
349                 while (nMotorEncoder(DISPENSER_MOT_PORT) > DISPENSER_POS2)
350                 {
351                         if(SensorValue[TOUCH_PORT])
352                         {
353                                 motor[DISPENSER_MOT_PORT] = 0;
354                                 stopAndKnock();
355                         }
356                 }
357                 motor[DISPENSER_MOT_PORT]= 0;
358
359                 drop_index = false;
360                 wait1Msec(100);
361
362                 // reset arm to initial position
363                 motor[DISPENSER_MOT_PORT] = -DISPENSER_SPEED;
364                 while (nMotorEncoder(DISPENSER_MOT_PORT) < DISPENSER_POS0)
365                 {
366                         if(SensorValue[TOUCH_PORT])
367                         {
368                                 motor[DISPENSER_MOT_PORT] = 0;
369                                 stopAndKnock();
370                         }
371                 }
372                 motor[DISPENSER_MOT_PORT] = 0;
373         }
374         openDoor();
375         domino_count--;
376 }
377
378 void somethingInTheWay (int motor_power) // Josh
379 {
380         // Stops motors, displays message and plays a sound. Exits when object is moved.
381         while(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
382         {
383                 setDriveTrainSpeed(0);
384                 eraseDisplay();
385                 displayString(5, "Please clear path ahead");
386                 playSound(soundBeepBeep);
387         }
388         ev3StopSound();
389         setDriveTrainSpeed(motor_power);
390 }
391
392 void somethingInTheWay (int left_mot_pow, int right_mot_pow)
393 {
394         // same as apove, just with 2 motor inputs to accomodate the
395         // use of this function in turns
396         while(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
```

```
397         {
398                 setDriveTrainSpeed(0);
399                 eraseDisplay();
400                 displayString(5, "Please clear path ahead");
401                 playSound(soundBeepBeep);
402         }
403         ev3StopSound();
404         motor[LEFT_MOT_PORT] = left_mot_pow;
405         motor[RIGHT_MOT_PORT] = right_mot_pow;
406 }
407
408 // ********************************* calculation functions ************************
409 int distToDeg(float dist)
410 {
411         // takes a distance and converts it to motor encoder clicks
412         // using wheel radius
413         return dist*180/PI/WHEEL_RAD;
414 }
415
416 float degToDist(int deg)
417 {
418         // converts degrees to motor encoder clicks using wheel radius
419         return deg*PI*WHEEL_RAD/180;
420 }
421
422 float average(int value1, int value2)
423 {
424         // returns average of two fucntions
425         return (abs(value1 + value2)/2.0);
426 }
427
428 // ********************************* movement functions ************************
429 void setDriveTrainSpeed(int speed)
430 {
431         // accomodates the backwards mounting of drive motors
432         motor[LEFT_MOT_PORT] = motor[RIGHT_MOT_PORT] = -1*speed;
433 }
434
435 void driveDist(float dist, int mot_pow)
436 {
437         // drives specified distance without dropping dominoes
438         setDriveTrainSpeed(mot_pow);
439         nMotorEncoder[LEFT_MOT_PORT] = 0;
440         while(abs(nMotorEncoder[LEFT_MOT_PORT]) < distToDeg(dist))
441         {
442                 // check for break conditions
443                 if(SensorValue[TOUCH_PORT])
444                 {
445                         stopAndKnock();
446                 }
447                 else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
448                 {
449                         somethingInTheWay(mot_pow);
450                 }
451         }
452         setDriveTrainSpeed(0);
453 }
```

```
454
455 void driveWhileDropping(float dist, int mot_pow, bool &drop_index, int &domino_count, float
    &dist_since_last_dom)
456 {
457         // drives specified distance while dropping dominos at consistent intervals
458         setDriveTrainSpeed(mot_pow);
459         nMotorEncoder[LEFT_MOT_PORT] = 0;
460         nMotorEncoder[RIGHT_MOT_PORT] = 0;
461         while(degToDist(abs(nMotorEncoder(LEFT_MOT_PORT))) < dist && domino_count > 0)
462         {
463                 // check for break conditions
464                 if(SensorValue[TOUCH_PORT])
465                 {
466                         stopAndKnock();
467                 }
468                 else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
469                 {
470                         somethingInTheWay(mot_pow);
471                 }
472
473                 // drop domino every DIST_BETWEEN_DOMINOS
474                 if(degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT))) + dist_since_last_dom >=
    DIST_BETWEEN_DOMINOS)
475                 {
476                         dist_since_last_dom = 0;
477                         nMotorEncoder(RIGHT_MOT_PORT) = 0;
478                         dropDomino(drop_index, domino_count);
479                         setDriveTrainSpeed(mot_pow);
480                 }
481         }
482         // allows for a smooth transition in the domino path between driving linearly and
    turning
483         dist_since_last_dom = degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT)));
484 }
485
486 void turnInPlace(int angle, int mot_pow)
487 {
488         int initialGyro = getGyroDegrees(GYRO_PORT);
489         if(angle < 0)
490         {
491                 // turn left
492                 motor[LEFT_MOT_PORT] = mot_pow;
493                 motor[RIGHT_MOT_PORT] = -mot_pow;
494                 while(getGyroDegrees(GYRO_PORT) > initialGyro+angle)
495                 {
496                         // check for break conditions
497                         if(SensorValue[TOUCH_PORT])
498                         {
499                                 stopAndKnock();
500                         }
501                         else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
502                         {
503                                 somethingInTheWay(mot_pow, -mot_pow);
504                         }
505                 }
506         }
507         else if(angle > 0)
508         {
```

```
509                 // turn right
510                 motor[LEFT_MOT_PORT] = -mot_pow;
511                 motor[RIGHT_MOT_PORT] = mot_pow;
512                 while(getGyroDegrees(GYRO_PORT) < initialGyro+angle)
513                 {
514                         // check for break conditions
515                         if(SensorValue[TOUCH_PORT])
516                         {
517                                 stopAndKnock();
518                         }
519                         else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
520                         {
521                                 somethingInTheWay(-mot_pow, mot_pow);
522                         }
523                 }
524 }
525
526         setDriveTrainSpeed(0);
527 }
528
529 void turnWhileDropping(int angle, int speed, bool &drop_index, int &domino_count, float
    &dist_since_last_dom)
530 {
531         // some concepts taken from:
532         // https://math.stackexchange.com/questions/4310012/calculate-the-turning-radius-
    turning-circle-of-a-two-wheeled-car
533
534         // turns the robot through a specific radius while dropping dominoes
535         float const TURN_RATIO = (TURN_RAD-13.5)/TURN_RAD;
536         int initialGyro = getGyroDegrees(GYRO_PORT);
537         if(angle > 0)
538         {
539                 // turn Right
540                 motor[LEFT_MOT_PORT] = -speed;
541                 motor[RIGHT_MOT_PORT] = -speed*TURN_RATIO;
542                 nMotorEncoder(LEFT_MOT_PORT) = 0;
543                 while(getGyroDegrees(GYRO_PORT) < initialGyro+angle && domino_count > 0)
544                 {
545                         // check for break conditions
546                         if(SensorValue[TOUCH_PORT])
547                         {
548                                 stopAndKnock();
549                         }
550                         else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
551                         {
552                                 somethingInTheWay(-speed, -speed*TURN_RATIO);
553                         }
554
555                         if(degToDist(abs(nMotorEncoder(LEFT_MOT_PORT))) + dist_since_last_dom
    >= DIST_BET_DOM_TURNING)
556                         {
557                                 // drops domino if correct spacing is reached
558                                 dist_since_last_dom = 0;
559                                 nMotorEncoder(LEFT_MOT_PORT) = 0;
560                                 dropDomino(drop_index, domino_count);
561                                 motor[LEFT_MOT_PORT] = -speed;
562                                 motor[RIGHT_MOT_PORT] = -speed*TURN_RATIO;
563                         }
```

```
564                 }
565                 dist_since_last_dom = degToDist(abs(nMotorEncoder(LEFT_MOT_PORT)));
566         }
567         else if(angle < 0)
568         {
569                 // turn left
570                 motor[LEFT_MOT_PORT] = -speed*TURN_RATIO;
571                 motor[RIGHT_MOT_PORT] = -speed;
572                 nMotorEncoder(RIGHT_MOT_PORT) = 0;
573                 while(getGyroDegrees(GYRO_PORT) > initialGyro+angle && domino_count > 0)
574                 {
575                         // check for break conditions
576                         if(SensorValue[TOUCH_PORT])
577                         {
578                                 stopAndKnock();
579                         }
580                         else if(SensorValue[ULTRASONIC_PORT] < DIST_IN_FRONT_LIM)
581                         {
582                                 somethingInTheWay(-speed*TURN_RATIO, -speed);
583                         }
584                         if(degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT))) +
dist_since_last_dom >= DIST_BET_DOM_TURNING)
585                         {
586                                 // drops domino if correct spacing is reached
587                                 dist_since_last_dom = 0;
588                                 nMotorEncoder(RIGHT_MOT_PORT) = 0;
589                                 dropDomino(drop_index, domino_count);
590                                 motor[LEFT_MOT_PORT] = -speed*TURN_RATIO;
591                                 motor[RIGHT_MOT_PORT] = -speed;
592                         }
593                 }
594                 dist_since_last_dom = degToDist(abs(nMotorEncoder(RIGHT_MOT_PORT)));
595         }
596 }
597
598 void stopAndKnock() // Josh
599 {
600         // moves backwards, knocking over first domino
601         nMotorEncoder(LEFT_MOT_PORT) = 0;
602         setDriveTrainSpeed(KNOCK_SPEED);
603         while(nMotorEncoder(LEFT_MOT_PORT) < distToDeg(DIST_BETWEEN_DOMINOS-0.5))
604         {}
605         setDriveTrainSpeed(0);
606         stopAllTasks();
607 }
608
609 void openDoor() // Henrique
610 {
611         motor[DOOR_MOT_PORT] = DOOR_SPEED;
612         while (nMotorEncoder(DOOR_MOT_PORT)<DOOR_ANG)
613         {
614                 // check for break conditions
615                 if(SensorValue[TOUCH_PORT])
616                 {
617                         motor[DOOR_MOT_PORT] = 0;
618                         stopAndKnock();
619                 }
620         }
```

```
621         motor[DOOR_MOT_PORT] = 0;
622 }
623
624 void closeDoor() // Henrique
625 {
626         if(!nMotorEncoder(DOOR_MOT_PORT)<5)
627         {
628                 motor[DOOR_MOT_PORT] = -1*DOOR_SPEED;
629                 while (nMotorEncoder(DOOR_MOT_PORT)>5)
630                 {
631                         // check for break conditions
632                         if(SensorValue[TOUCH_PORT])
633                         {
634                                 motor[DOOR_MOT_PORT] = 0;
635                                 stopAndKnock();
636                         }
637                 }
638                 motor[DOOR_MOT_PORT] = 0;
639         }
640 }
```