/\* Control Z: 7

Castling short: 8

Castling long: 9

Super User: 10

Calibrate: 11

Wondering: 12 \*/

/\* WAT ER NOG MOET GEBEUREN:

VANAF KING: CONTROLEER OF KONING SCHAAK STAAT, OP DEZELFDE MANIER ALS PAWN, KNIGHT EN BISSHOP.

VERDER: ZORG ERVOOR DAT HET SCHAAKGEVENDE STUK GESLAGEN KAN WORDEN.

DIT DOOR MIDDEL VAN TIJDELIJK UPDATEN, DAN CONTROLEREN, ALS HET NIET KAN --> TERUGVERANDEREN!

\*/

/\* How to set up the pieces: white king should be on E1, black king on E8 \*/

//Declaring variables

//Changeable constants:

byte resetnumber = 20;

int Speed = 400; // the lower the faster

byte magnethigh = 68; //degreesaut

byte magnetlow = 30;

unsigned int delaytime = 5000; // number of milliseconds to wait after first part of voice contol has been said

// set pins for stepper motor drivers

byte dirpinx = 2; // determines the direction of motor 1

byte steppinx = 3; // determines the number of steps of motor 1

byte dirpiny = 4; // determines the direction of motor 2

byte steppiny = 5; // determines the number of sfteps of motor 2

boolean knight = 0; //if a knight moves or if it is another piece

boolean turn = 0; //0:whit's turn, 1 black's turn

boolean dirx; // direction of the motor that controls x-movement

boolean diry; // direction of the motor that controls y-movement

byte resetx = digitalRead(7); // resetbutton (pin 7). Turns 0 when pressed

byte resety = digitalRead(8); // resetbutton (pin 8). Turns 0 when pressed

byte x1 = 0; // starting x

byte y1 = 0; // starting y

boolean slain; // slain or not

byte x2 = 0; // finished x

byte y2 = 0; // finished y

byte flow; // has to do with program flow. It can be 0 (don't no move done), 1 (allow a move to be done), 2 (special kind of voice control) or 3 (multiple possible moves, say beginning position)

byte j = 0; // for switch command

byte x3; // current x position

byte y3; // current y position

byte xrest; // x coordinate of the rest square (after capturerun)

byte yrest; // y coordinate of the rest square (after capturerun)

byte enpassant [] = {0, 0}; // 0= white file double move, 1= black file double move

byte promote = 0; // 0 = untrue, 1 = true, 2 = knight, 3 = bishop, 4 = rook, 5 = queen

byte nextxreset = resetnumber; // every 5 moves, the position should be calibrated at least once

byte nextyreset = resetnumber;

// location to move the slain pieces

//byte slainposx [] = {10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 8, 7, 6, 5, 4, 3, 2, 1, 8, 7, 6, 5, 4, 3, 2, 1}; // x coordinates of white graveyard pieces

//byte slainposy [] = {6, 5, 4, 7, 3, 8, 2, 9, 1, 10, 5, 6, 4, 7, 3, 8, 2, 9, 1, 10, 10, 10, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9, 9, 9}; // y coordinates of white graveyard pieces

byte slainpos1 [] = {10, 10, 10, 10, 10, 10, 10, 10, 10, 9, 9, 9, 9, 9, 9}; // (slainpos1[0],slainpos2[0]) is (10,5), which is a white grave field (d:) Used this way because of the symmetry of the board. White: (1,2). Black (2,1)

byte slainpos2 [] = {5, 4, 6, 3, 7, 2, 8, 1, 9, 5, 4, 6, 3, 7, 2}; // (slainpos2[0],slainpos1[0]) is (5,10), which is a black grave field (?5)

byte totalslainw = 0;

byte totalslainb = 0;

boolean resetxmove = LOW; // if rank '1' selected

boolean resetymove = LOW; // if file 'H' selected

byte type = 0;// stores what type of piece a piece is

unsigned long counter;

boolean superuser = false; // absolute command, no rules :)

/\* A1 = [1,8] H1 = [1,1] A8 = [8,8] H8 = [8,1] empty=0, out of bounds=13 Out of bounds is om ervoor te zorgen dat

white pawn=1, wknight=2, wbishop=3, wrook=4, wqueen=5, wking=6,

black pawn=7, bknight=8, bbishop=9, brook=10, bqueen=11, bking=12, \*/

byte king[2][2] = {

{3,4}, // coordinates white king

{8,4} // coordinates black king

};

byte CB[12][12] = {

{ 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20},

{ 20, 4, 2, 3, 0, 5, 3, 2, 4, 0, 0, 20},

{ 20, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 20},

{ 20, 0, 0, 0, 6, 0, 0, 0, 10, 0, 0, 20},

{ 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20},

{ 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20},

{ 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20},

{ 20, 7, 7, 7, 7, 7, 7, 7, 7, 0, 0, 20},

{ 20, 10, 8, 9, 12, 11, 9, 8, 0, 0, 0, 20},

{ 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20},

{ 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 20},

{ 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20}

};

//Imports the librarys to the sketch

#include <BitVoicer11.h>

#include <Servo.h>

Servo myservo;

BitVoicerSerial bvSerial = BitVoicerSerial();

byte dataType = 0;

byte g = 0; // inspringen van seriele monitor

// declaring global variables

void setup()

{

Serial.begin(9600);

resetx = 1;

resety = 1;

pinMode(dirpinx, OUTPUT); // x motor, direction

pinMode(steppinx, OUTPUT); // x motor, number of steps

pinMode(dirpiny, OUTPUT); //y motor

pinMode(steppiny, OUTPUT);

myservo.attach(6); // servo

pinMode(7, INPUT\_PULLUP); // x resetbutton

pinMode(8, INPUT\_PULLUP); // y resetbutton

pinMode(9, OUTPUT); //blue led, turn low to turn on

pinMode(10, OUTPUT); //green led, turn low to turn on

pinMode(11, OUTPUT); // red led, turn low to turn on

pinMode(12, OUTPUT); // white turn light

digitalWrite(12, HIGH); // white starts

pinMode(13, OUTPUT); // black turn light

magnet (1, HIGH); // make the magnet start up

reset(1); // put the magnet in starting position

feedbacklight(1,0); // turn the feedback light off.

} // 1

void loop()

{

if ((j == 2 || flow == 3 || type == 10 || promote == 1) && millis() > (counter + 5000)) {

j = 0;

type = 0;

feedbacklight(1,0);

flow = 0;

g = 0;

} // reset if nothing has happend

/\*if (Serial.available() > 0) {

delay(10); // this delay is necessary for the data streaming

serialEvent2(2);

} // if there is a character to be read \*/

} // 2

byte decoder(byte k)

{

char arg[] = {k,127}; a(3, arg);

dataType = bvSerial.getData(); // 4 = string (for example "b4") 2 = int (for example 2 (knight))

byte x1t = x1, x2t = x2, y1t = y1, y2t = y2; // temporary variables, to keep control Z intact

if(dataType == 2) {

byte previousType = type;

type = bvSerial.intData; // type can be 1 (pawn), 2 (knight), 3 (bishop), 4 (rook), 5 (queen), 6 (king), 7 (castling kingside), 8 (castling queenside), 9 (Control Z), 10 (super user), 11 (calibrate) or 12 (wondering)

if(type == 7 && (previousType == 7 || x1 == 0)) {

feedbacklight(3,2); // red light

g--; return 1;

} // control Z used multiple times in a row

if (promote == 1) {

if (type > 1 && type < 6) {

promote = type;

CB[x2][y2] - promote; // update matrix

CB[x1][y1] = 0;

feedbacklight(3,0);

flow = 1;

g--; return 1;

} // 2, 3, 4 or 5

else {

feedbacklight(3,2);

type = 0;

j = 0;

promote = 0;

g--; return 1;

} // invalid promotion input

}

if (type != 7 && type != 8 && type != 9) {

counter = millis(); // start counting

feedbacklight(3,1); // turn light green

j = 2; // j=2 and j=3 regulate the final coordinates

} // 7, 8 and 9 are stand-alone commands

if (type == 7) {

static boolean lastZ = false; // for using repetitive Control-z's

byte a = x1, b = y1; // store x1, b stores y1

x1 = x2;

y1 = y2;

x2 = a;

y2 = b;

boolean r = false; // rembembers in there was an en passant capture

if(enpassant[1-turn] == 9) r = true;

else r = false;

zet(3);

if (r) enpassant[turn] = 9;

CB[x2][y2] = CB[x1][y1]; // for example "Queen to A4 makes (1,4) = 5

CB[x1][y1] = 0; // and the place where the queen stood becomes 0

if (slain || enpassant[turn] == 9) {

if (turn) {

magnet(3,LOW);

movem(3,(slainpos1[totalslainw-1]-x3) \* 2, (slainpos2[totalslainw-1]-y3) \* 2);

x3 = slainpos1[totalslainw-1]; y3 = slainpos2[totalslainw-1];

magnet(3,HIGH);

if (enpassant[turn] == 9) {

x1 = 4; // put white piece at rank 4

enpassant[1 - turn] = y1;

enpassant[turn] = 0;

}

}

else {

magnet(3,LOW);

movem(3,(slainpos2[totalslainb-1]-x3) \* 2, (slainpos1[totalslainb-1]-y3) \* 2);

x3 = slainpos2[totalslainb-1]; y3 = slainpos1[totalslainb-1];

magnet(3,HIGH);

if (enpassant[turn] == 9) {

x1 = 5; // put black piece at rank 5

enpassant[1 - turn] = y1;

enpassant[turn] = 0;

}

}

if (x1 - x3 == 0) {

movem(3,1, (y1 - y3) / abs(y1 - y3)); // half a field x, half a field y

movem(3,0, (y1 - y3 - 1 \* (y1 - y3) / abs(y1 - y3)) \* 2); // half a square less than the required distance

movem(3,-1, (y1 - y3) / abs(y1 - y3)); // move half a field diagnoally

} // if 'grave' square has same x coordinate as slain piece // From (x3,y3) to (x1,y1)

if (y1 - y3 == 0) {

movem(3,(x1 - x3) / abs(x1 - x3), 1);

movem(3,(x1 - x3 - 1 \* (x1 - x3) / abs(x1 - x3)) \* 2, 0);

movem(3,(x1 - x3) / abs(x1 - x3), -1);

} // if grave square has same y coordinate as slain piece, see x

if (x1 - x3 != 0 && y1 - y3 != 0) {

movem(3,(x1 - x3) / abs(x1 - x3), (y1 - y3) / abs(y1 - y3)); //to make sure it goes in the right direction, half a field in the x and half a field in the y direction

movem(3,0, (y1 - y3 - 1 \* (y1 - y3) / abs(y1 - y3)) \* 2); // half a square less that required distance, y

movem(3,(x1 - x3 - 1 \* (x1 - x3) / abs(x1 - x3)) \* 2, 0); // half a square less than required distance, x

movem(3,(x1 - x3) / abs(x1 - x3), (y1 - y3) / abs(y1 - y3)); // final half a diagonal field.

} // not same x coordinate, not same y coordinate

x3 = x1; y3 = y1; // update coordinates

magnet(3,LOW);

if (turn) {

CB[x1][y1] = CB[slainpos1[totalslainw - 1]][slainpos2[totalslainw - 1]];

CB[slainpos1[totalslainw - 1]][slainpos2[totalslainw - 1]] = 0;

totalslainw--;

}

else {

CB[x1][y1] = CB[slainpos2[totalslainb - 1]][slainpos1[totalslainb - 1]];

CB[slainpos2[totalslainb - 1]][slainpos1[totalslainb - 1]] = 0;

totalslainb--;

}

}

lastZ = 1 - lastZ;

} // Control Z

if (type == 8) {

if(CB[1 + 7 \* turn][4] == 6 + 6 \* turn && CB[1 + 7 \* turn][3] == 0 && CB[1 + 7 \* turn][2] == 0 && CB[1 + 7 \* turn][1] == 4 + 6 \* turn) {

magnet(3,LOW);

movem(3,(1 + 7 \* turn - x3) \* 2, (1 - y3) \* 2); // move to rook

magnet(3,HIGH);

movem(3,1 - 2 \* turn , 3); // move rook

magnet(3,LOW);

movem(3,-1 + 2 \* turn, 3); // move to king

magnet(3,HIGH);

movem(3,0, -4); // move king

magnet(3,LOW);

movem(3,1 - 2 \* turn , 1); // move to rook

magnet(3,HIGH);

movem(3,-1 + 2 \* turn, 1); // move rook

x3 = 1 + 7 \* turn;

y3 = 3;

CB[1 + 7 \* turn][1] = 0; // location where the rook stood

CB[1 + 7 \* turn][2] = 6 + 6 \* turn ; // new location of the king

CB[1 + 7 \* turn][3] = 4 + 6 \* turn; // new location of the rook

CB[1 + 7 \* turn][4] = 0; // location where the king stood

king[turn][1] = 2; // update king (y coordinate)

switchturn(3);

} // check if possible

else

feedbacklight(3,2); // red light

} // castling short

if (type == 9) {

if(CB[1 + 7 \* turn][4] == 6 + 6 \* turn && CB[1 + 7 \* turn][5] == 0 && CB[1 + 7 \* turn][6] == 0 && CB[1 + 7 \* turn][7] == 0 && CB[1 + 7 \* turn][8] == 4 + 6 \* turn) {

magnet(3,LOW);

movem(3,(1 + 7 \* turn - x3) \* 2, (8 - y3) \* 2); // move to rook

magnet(3,HIGH);

movem(3,1 - 2 \* turn, -5); // move rook

magnet(3,LOW);

movem(3,-1 + 2 \* turn, -3); // move to king

magnet(3,HIGH);

movem(3,0, 4); // move king

magnet(3,LOW);

movem(3,1 - 2 \* turn, -1); // move to rook

magnet(3,HIGH);

movem(3,-1 + 2 \* turn, -1); // move rook\

x3 = 1 + 7 \* turn;

y3 = 5;

CB[1 + 7 \* turn][4] = 0; // location where the king stood

CB[1 + 7 \* turn][5] = 4 + 6 \* turn ; // new location of the rook

CB[1 + 7 \* turn][6] = 6 + 6 \* turn; // new location of the king

CB[1 + 7 \* turn][8] = 0; // location where the rook stood

king[turn][1] = 2; // update king (y coordinate)

switchturn(3);

} // check if possible

else

feedbacklight(3,2); // flash red light

} // castling long

if (type == 10) {

feedbacklight(3,4); // purple

j = 0; // start with first coordinate

} // super user

if (type == 11) {

magnet(3,LOW);

reset(3);

magnet(3,HIGH);

} // calibrate

g--; return 1;

} //If number. The first character read is a number --> a type should be set.

if((dataType == 4) {

char field[1];

field[0] = in;

if(Serial.available() == 1 ) {

in = Serial.read();

if(in > 48 && in < 59) {

field[1] = in;

} // second character (number)

else {

// Serial.println("Invalid input.");

feedbacklight(3,2);

g--; return 1;

} // wrong second character

} // 2 characters total

else {

// Serial.println("Invalid input.");

feedbacklight(3,2);

while (Serial.available() > 0) {

in = Serial.read();

} // empty the serial buffer

g--; return 1;

} // one character, or more that 2 characters

for (byte LN = 0; LN < 2; LN++) {

byte in = field[LN]; //reads letter

switch (j) {

byte xw, yw;

case 0: // first character (first coordinate)

if (in > 62 && in < 73) {

in = in + 32;

} // translate uppercase to lowercase

if (in > 94 && in < 105) {

y1 = in - 96; // turn into 1-8

y1 = 9 - y1; // invert board

j = 1; // next case

} // if there is a lowercase a-h

break; //exit case

case 1: //second character (first coordinate)

if (in > 48 && in < 59) {

x1 = in - 48;

j = 2; // next case

if (flow == 3) {

print(3,true,x2,y2);

CB[x2][y2] = CB[x1][y1];

CB[x1][y1] = 0; // update matrix

feedbacklight(3,0); // turn off light

if (type == 1 && x2 == 8 - 7\*turn) {

promote = 1;

feedbacklight(3,4); // purple

counter = millis();

break;

}

flow = 1;

g--; return 1; // break out of the decoder() function

} // the beginning position has been given

counter = millis(); // start counting

if (type == 10) {

feedbacklight(3,1); // green light

} // super user, need more info!

} // if there is a ascii number (1-8)

else {

j = 0; // restart

x1 = x1t; y1 = y1t; x2 = x2t; y2 = y2t; // x1, y1, x2, y2 to their original value (for Control Z)

} // no ascii number

break;

case 2: // third character (second coordinate)

if (type == 10) {

feedbacklight(3,3); // blue light

} // super user

if (in > 62 && in < 73) {

in = in + 32;

} // turn into lowercase letter

if (in > 94 && in < 105) {

if (type == 12) {

yw = 105 - in;

j = 3;

break;

} // wondering

y2 = in - 96; // turn into number

y2 = 9 - y2; // invert axis

j = 3; // next case

} // if lowercase letter

else {

j = 0; // restart

x1 = x1t; y1 = y1t; x2 = x2t; y2 = y2t; // x1, y1, x2, y2 to their original value (for Control Z)

}

break; // exit case

case 3: // fourth character (second coordinate)

if (in > 48 && in < 59) {

if (type == 12) {

xw = in - 48;

j = 3;

print(3,false,xw,yw);

feedbacklight(3,0);

x1 = x1t; y1 = y1t; x2 = x2t; y2 = y2t; // x1, y1, x2, y2 to their original value (for Control Z)

g--; return 1;

} // wondering

x2 = in - 48; // turn into actual number

j = 0;

if (type == 10 || possible(3)) {

if (slain) {

if(turn) {

CB[slainpos1[totalslainw]][slainpos2[totalslainw]] = CB[x2][y2];

} // black's move --> white piece // update matrix grave square, maybe this should be in a different place

else {

CB[slainpos2[totalslainb]][slainpos1[totalslainb]] = CB[x2][y2];

} //

} // update matrix

if (flow == 3) {

counter = millis();

j = 0;

break; // break out of case 3

} // two possible moves

if (type == 1 && x2 == 8 - 7\*turn) {

promote = 1;

feedbacklight(3,4); // purple

counter = millis();

break;

} // promotion

print(3,true,x2,y2); // print the move

CB[x2][y2] = CB[x1][y1]; // update matrix: maybe this should be in a different place

CB[x1][y1] = 0;

if (type == 6) {

king[turn][0] = x2; king[turn][1] = y2;

} // update king

feedbacklight(3,0); // turn off light

flow = 1;

} // if possible() returns 1. Possible() can make flow = 3 if there are multiple possibilities) type == 10: super user!

else {

x1 = x1t; y1 = y1t; x2 = x2t; y2 = y2t; // x1, y1, x2, y2 to their original value (for Control Z)

} // not possible or super user, return values to their original

} // if ascii number (if valid)

else {

j = 0;

x1 = x1t; y1 = y1t; x2 = x2t; y2 = y2t; // x1, y1, x2, y2 to their original value (for Control Z)

} // if invalid (no ascii number)

break;

} // end switch

} // for loop, same as with voice control

} // lowercase letter or ? or @

g--;

} // 3

byte serialEvent(byte k) // when something has been typed in

{

char arg[] = {k,127}; a(4, arg);

decoder(4);

if (flow == 1) {

if (slain) {

capturerun(4); // sets attacking piece on rest square

trash(4,x2, y2); // moves slain piece to graveyard

} //checks if something is slain

zet(4); // does the actual move (and in case of slain, the final (third) part of the move)

} //If the input is a move, decoder sets flow to 1

g--;

} // 4

byte a(byte b,char d[]) // b is the number of the function,

{

/\*

String c;

switch (b) {

case 1: c = "setup("; break;

case 2: c = "loop("; break;

case 3: c = "decoder2("; break;

case 4: c = "serialEvent2("; break;

case 5: c = "autoreset("; break;

case 6: c = "capturerun("; break;

case 7: c = "checkpawn("; break;

case 8: c = "checkknight("; break;

case 9: c = "checkbishop("; break;

case 10: c = "checkrook("; break;

case 11: c = "checkqueen("; break;

case 12: c = "checkkking("; break;

case 13: c = "feedbacklight("; break;

case 14: c = "magnet("; break;

case 15: c = "movegrid("; break;

case 16: c = "movem("; break;

case 17: c = "possible("; break;

case 18: c = "print("; break;

case 19: c = "reset("; break;

case 20: c = "switchturn("; break;

case 21: c = "trash("; break;

case 22: c = "xreset("; break;

case 23: c = "yreset("; break;

case 24: c = "zet("; break;

case 25: c = "check("; break;

case 26: c = "checkpressure("; break;

}

for(byte i = 0; i < g; i++){

Serial.print(" ");

}

Serial.print(c);

for(byte i = 1; d[i] != 127; i++) {

if (i != 1) Serial.print(",");

Serial.print(d[i],10);

}

Serial.println(")");

g++;

\*/

}

byte autoreset(byte k, byte b) // number is 1 or 2, depending on which move it is. Autoreset occurs whenever the magnet goes to the H file or 1 rank

{

char arg[] = {k,b,127}; a(5,arg);

if (b == 1) {

if (x1 == 1) resetxmove = HIGH; // reset?

if (y1 == 1) resetymove = HIGH;

} // piece moves towards the button

else {

if (x2 == 1) {

resetxmove = HIGH;

} // reset?

if (y2 == 1) {

resetymove = HIGH;

}

} // piece moves away from the button

g--; // terminate

} // 5

byte capturerun(byte k)

{

char arg[] = {k,127}; a(6,arg);

if ((abs(x1 - x2) > 1 || abs(y1 - y2) > 1) && knight == 0) {

magnet(6,LOW);

movem(6,(x1 - x3) \* 2, (y1 - y3) \* 2); // to attacking piece

x3 = x1; y3 = y1;

magnet(6,HIGH);

movem(6,((x2 - x3) - (x2 - x3 != 0) \* (abs(x2 - x3) / (x2 - x3))) \* 2, ((y2 - y3) - (y2 - y3 != 0) \* (abs(y2 - y3) / (y2 - y3))) \* 2); // move one less than necessary

x3 = x3 + (x2 - x3) - (x2 - x3 != 0) \* (abs(x2 - x3) / (x2 - x3));

y3 = y3 + (y2 - y3) - (y2 - y3 != 0) \* (abs(y2 - y3) / (y2 - y3)); //update coordinates

xrest = x3; yrest = y3; // say that the attacking piece is at the rest square

} // more than 1 step, and no knight

g--;

} // 6

byte check(byte k)

{

char arg[] = {k,127}; a(25,arg);

if (type == 6){

if(checkpressure(25,x2,y2)) {

g--; return 1;

} // king has moved

}

else {

if(checkpressure(25,king[turn][0],king[turn][1])) {

g--; return 1;

} // king has not moved

}

g--; return 0;

} // 25

byte checkpressure (byte k,byte x, byte y)

{

char arg[] = {k,x,y,127}; a(26,arg);

if(checkpawn(26,x,y,1-turn) || checkknight(26,x,y,1-turn) || checkbishop(26,x,y,1-turn) || checkrook(26,x,y,1-turn) || checkqueen(26,x,y,1-turn) /\*|| checkking(26,x,y)\*/) {

g--; return 1;

}

else

g--; return 0;

} // 26

byte checkpawn(byte k, byte x, byte y, byte t)

{

char arg[] = {k,x,y,t,127}; a(7,arg);

byte a = 0; // possibilities

if ((k == 26 || slain == 1) && (CB[x - 1 + 2 \* t][y - 1] == 1 + 6 \* t)) {

if (k == 26) {

g--; return 1;

} // checkpressure

a++;

x1 = x2 - 1 + 2 \* t;

y1 = y2 - 1;

} // diagonal capture from lower file

if ((k == 26 || slain == 1) && (CB[x - 1 + 2 \* t][y + 1] == 1 + 6 \* t)) {

if (k == 26) {

g--; return 1;

} // checkpressure

a++;

x1 = x2 - 1 + 2 \* t;

y1 = y2 + 1;

} // diagonal capture from higher file

if (slain == 0 && CB[x - 1 + 2 \* t][y] == 1 + 6 \* t) {

if (k == 26) {

g--; return 0;

}

a++;

x1 = x2 - 1 + 2 \* t;

y1 = y2;

} // move one field

if (x == 4 + t && CB[2 + 5 \* t][y] == 1 + 6 \* t && CB[3 + 3 \* t][y] == 0 && CB[x][y] == 0) {

if (k == 26) {

g--; return 0;

}

a ++;

x1 = x2 - 2 + 4 \* t;

y1 = y2;

enpassant[t] = y1; // indicates what file the pawn is in

} // move two fields

if(x == 6 - 3 \* t && y == enpassant[1 - t]) {

if (k == 26) {

g--; return 0;

}

if (CB[5 - t][y2 - 1] == 1 + 6 \* t ) {

a++;

x1 = 5 - t;

y1 = y2 - 1;

enpassant[t] = 9; // give it a special value so that it can be recalled later

} // en passant from higher file

if (CB[5 - t][y2 + 1] == 1 + 6 \* t ) {

a++;

x1 = 5 - t;

y1 = y2 + 1;

enpassant[t] = 9; // give it a special value so that it can be recalled later

} // en passant from lower file

} // en passant

g--; return a;

} // 7

byte checkknight(byte k, byte x, byte y, byte t)

{

char arg[] = {k,x,y,t,127}; a(8,arg);

byte a = 0; // possibilities

if (x - 2 > 0) {

if (CB[x - 2][y - 1] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a ++;

x1 = x2 - 2;

y1 = y2 - 1;

} // from higher file

if (CB[x - 2][y + 1] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a ++;

x1 = x2 - 2;

y1 = y2 + 1;

} // from lower file

} // not off edge, two ranks lower

if (x + 2 < 9) {

if (CB[x + 2][y - 1] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a ++;

x1 = x2 + 2;

y1 = y2 - 1;

} // from higher file

if (CB[x + 2][y + 1] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a++;

x1 = x2 + 2;

y1 = y2 + 1;

} // from lower file

} // not off edge, two ranks higher

if (y - 2 > 0) {

if (CB[x - 1][y - 2] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a ++;

x1 = x2 - 1;

y1 = y2 - 2;

} // from lower rank

if (CB[x + 1][y - 2] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a++;

x1 = x2 + 1;

y1 = y2 - 2;

} // from higher rank

} // not off edge, two files higher

if (y + 2 < 9) {

if (CB[x - 1][y + 2] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a ++;

x1 = x2 - 1;

y1 = y2 + 2;

} // from lower rank

if (CB[x + 1][y + 2] == 2 + 6 \* t) {

if (k == 26) {

g--; return 1;

}

a++;

x1 = x2 + 1;

y1 = y2 + 2;

} // from higher rank

} // not off edge, two files lower

g--; return a;

} // 8

byte checkbishop(byte k, byte x, byte y, byte t)

{

char arg[] = {k,x,y,t,127}; a(9,arg);

byte c = 0; // possibilities

char a = 0, b = 0;

do {

a++;

b++;

if (CB[x + a][y + b] == 3 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2 + a;

y1 = y2 + b;

}

} while (CB[x + a][y + b] == 0); // check right top side

a = 0; b = 0;

do {

a--;

b++;

if (CB[x + a][y + b] == 3 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2 + a;

y1 = y2 + b;

}

} while (CB[x + a][y + b] == 0); // check left top side

a = 0; b = 0;

do {

a++;

b--;

if (CB[x + a][y + b] == 3 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2 + a;

y1 = y2 + b;

}

} while (CB[x + a][y + b] == 0); // check right bottom side

a = 0; b = 0;

do {

a--;

b--;

if (CB[x + a][y + b] == 3 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2 + a;

y1 = y2 + b;

}

} while (CB[x + a][y + b] == 0); // check left bottom side

a = 0; b = 0;

g--; return c;

} // 9

byte checkrook(byte k, byte x, byte y, byte t)

{

char arg[] = {k,x,y,t,127}; a(10,arg);

byte c = 0; // possibilites

char a = 0, b = 0;

do {

a++;

if (CB[x + a][y] == 4 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2 + a;

y1 = y2;

}

} while (CB[x + a][y] == 0); // check right side

a = 0;

do {

a--;

if (CB[x + a][y] == 4 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2 + a;

y1 = y2;

}

} while (CB[x + a][y] == 0); // check left side

a = 0;

do {

b++;

if (CB[x][y + b] == 4 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2;

y1 = y2 + b;

}

} while (CB[x][y + b] == 0); // check top side

b = 0;

do {

b--;

if (CB[x][y + b] == 4 + 6 \* t && x + a < 9 && y + b < 9) {

if (k == 26) {

g--; return 1;

}

c++;

x1 = x2;

y1 = y2 + b;

}

} while (CB[x][y + b] == 0); // check bottom side

b = 0;

g--; return c;

} // 10

byte checkqueen(byte k, byte x, byte y, byte t)

{

char arg[] = {k,x,y,t,127}; a(11,arg);

byte c = 0; // possibilites

char a = 0, b = 0;

do {

a++;

b++;

if (CB[x + a][y + b] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2 + a;

y1 = y2 + b;

}

} while (CB[x + a][y + b] == 0); // check right top side

a = 0; b = 0;

do {

a--;

b++; // check left top side

if (CB[x + a][y + b] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2 + a;

y1 = y2 + b;

} // queen

} while (CB[x + a][y + b] == 0); // check right top side

a = 0; b = 0;

do {

a++;

b--;

if (CB[x + a][y + b] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2 + a;

y1 = y2 + b;

}

} while (CB[x + a][y + b] == 0); // check right bottom side

a = 0; b = 0;

do {

a--;

b--;

if (CB[x + a][y + b] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2 + a;

y1 = y2 + b;

}

} while (CB[x + a][y + b] == 0); // check left bottom side

a = 0; b = 0;

do {

a++;

if (CB[x + a][y] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2 + a;

y1 = y2;

}

} while (CB[x + a][y] == 0); // check right side

a = 0;

do {

a--;

if (CB[x + a][y] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2 + a;

y1 = y2;

}

} while (CB[x + a][y] == 0); // check left side

a = 0;

do {

b++;

if (CB[x][y + b] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2;

y1 = y2 + b;

}

} while (CB[x][y + b] == 0); // check top side

b = 0;

do {

b--;

if (CB[x][y + b] == 5 + 6 \* t && x + a < 9 && y + b < 9) {

c++;

x1 = x2;

y1 = y2 + b;

}

} while (CB[x][y + b] == 0); // check bottom side

b = 0;

g--; return c;

} // 11

byte checkking(byte k, byte x, byte y)

{

char arg[] = {k,127}; a(12,arg);

byte c = 0; // possibilties

if (CB[x2 + 1][y2 + 1] == 6 + 6 \* turn) {

c++;

x1 = x2 + 1;

y1 = y2 + 1;

} // check top right

if (CB[x2 - 1][y2 + 1] == 6 + 6 \* turn) {

c++;

x1 = x2 - 1;

y1 = y2 + 1;

} // check top left

if (CB[x2 + 1][y2 - 1] == 6 + 6 \* turn) {

c++;

x1 = x2 + 1;

y1 = y2 - 1;

} // check bottom right

if (CB[x2 - 1][y2 - 1] == 6 + 6 \* turn) {

c++;

x1 = x2 - 1;

y1 = y2 - 1;

} // check bottom left

if (CB[x2 + 1][y2] == 6 + 6 \* turn) {

c++;

x1 = x2 + 1;

y1 = y2;

} // check right

if (CB[x2 - 1][y2] == 6 + 6 \* turn) {

c++;

x1 = x2 - 1;

y1 = y2;

} // check left

if (CB[x2][y2 + 1] == 6 + 6 \* turn) {

c++;

x1 = x2;

y1 = y2 + 1;

} // check top

if (CB[x2][y2 - 1] == 6 + 6 \* turn) {

c++;

x1 = x2;

y1 = y2 - 1;

} // check right

g--; return c;

} // 12

byte feedbacklight(byte k, byte z)

{

char arg[] = {k,z,127}; a(13,arg);

digitalWrite(9,HIGH);

digitalWrite(10,HIGH);

digitalWrite(11,HIGH);

delay(50);

if (z == 0) {

digitalWrite(9, HIGH);

digitalWrite(10, HIGH);

digitalWrite(11, HIGH);

} // off

if (z == 1) {

digitalWrite(9, HIGH);

digitalWrite(10, LOW);

digitalWrite(11, HIGH);

} // green

if (z == 2) {

digitalWrite(9, HIGH);

digitalWrite(10, HIGH);

digitalWrite(11, LOW);

delay(1000); // wait a second

digitalWrite(11, HIGH); // turn off again

} // red

if (z == 3) {

digitalWrite(9, LOW);

digitalWrite(10, HIGH);

digitalWrite(11, HIGH);

} // blue

if (z == 4) {

digitalWrite(9,LOW);

digitalWrite(10,HIGH);

digitalWrite(11,LOW);

} // purple

g--;

} // 13

byte magnet(byte k, boolean z)

{

char arg[] = {k,char(z),127}; a(14,arg);

if (z == LOW) {

myservo.write(magnetlow);

} // low position

else {

myservo.write(magnethigh); // high position

}

delay(200); // give the servo some time before the motors start

g--;

} // 14

byte movegrid(byte k, byte x, byte y) // moves from the current position to (x,y) over the empty grid

{

char arg[] = {k,x,y,127}; a(15,arg);

// White: (1,2). Black (2,1). Slainpos2[totalslainw] = y, slainpos1[totalslainw] = x, slainpos2[totalslainb] = x, slainpos1[totalslainb] = y

if (x - x3 == 0) {

movem(15,-1 + 2\*(x < 5), (y - y3) / abs(y - y3)); // half a field x (1 for x<5, -1 for x>4), half a field y

movem(15,0, (y - y3 - 1 \* (y - y3) / abs(y - y3)) \* 2); // half a square less than the required distance

movem(15,1 - 2\*(x < 5), (y - y3) / abs(y - y3)); // move half a field diagnoally

} // same x-coordinate (same rank)

if (y - y3 == 0) {

movem(15,(x - x3) / abs(x - x3), -1 + 2\*(y < 5)); // half a field x, half a field y

movem(15,(x - x3 - 1 \* (x - x3) / abs(x - x3)) \* 2, 0); // half a square less than the required distance

movem(15,(x - x3) / abs(x - x3), 1 - 2\*(y < 5));

} // same y-coordinate (same file)

if (x - x3 != 0 && y - y3!= 0) {

movem(15,(x - x3) / abs(x - x3), (y - y3) / abs(y - y3)); //to make sure it goes in the right direction, half a field in the x and half a field in the y direction

movem(15,(x - x3 - 1 \* (x - x3) / abs(x - x3)) \* 2, 0); // half a square less than required distance, x

movem(15,0, (y - y3 - 1 \* (y - y3) / abs(y - y3)) \* 2); // half a square less that required distance, y

movem(15,(x - x3) / abs(x - x3), (y - y3) / abs(y - y3)); // final half a diagonal field.

} // different rank, different file

x3 = x; y3 = y; // update coordinates

magnet(15,LOW);

g--;

} // 15

byte movem(byte k, int x, int y)

{

char arg[] = {k,x,y,127}; a(16,arg);

nextxreset--; // count down number of reset moves

nextyreset--;

//determine direction

if (x < 0) {

digitalWrite(dirpinx, HIGH); // negative x direction on our board

dirx = LOW; // variable that indecates the x direction

} //by changing '<' to '>' an axis direction can be changed

else {

digitalWrite(dirpinx, LOW); // positive x direction

dirx = HIGH;

}

if (y > 0) {

digitalWrite(dirpiny, HIGH); // positive y direction

diry = HIGH; //variable that indicates the y direction

}

else {

digitalWrite(dirpiny, LOW); // negatibe y direction

diry = LOW;

}

// execute the steps

int square = 520; // number of steps for one field

x = abs(x \* square); // only a positive number of steps can be executed, so x fields

y = abs(y \* square); // y fields

for (int i = 0; i < max(x, y); i++) {

if (i < x ) {

int resetx = digitalRead(7); // resetbutton on the x axis, is LOW when pressed

if (resetx == HIGH || dirx == HIGH) {

digitalWrite(steppinx, LOW);

digitalWrite(steppinx, HIGH); //This change from LOW to HIGH is necessary for the execution of the stepper.

delayMicroseconds(Speed); // 500 microseconds delay between each step, 990 steps for one square meßans half a second per field. In theory.

} // EITHER the button is not pressed, OR the magnet moves away from the button (positive x direction)

}

if (i < y) {

int resety = digitalRead(8);

if (resety == HIGH || diry == HIGH) {

digitalWrite(steppiny, LOW);

digitalWrite(steppiny, HIGH);

delayMicroseconds(Speed);

}

} // same story for the y direction

} // the loop continues until the highest value of x and y is reached

if (resetxmove == HIGH) {

xreset(16);

}

if (resetymove == HIGH) {

yreset(16);

}

g--;

} // 16

byte possible(byte k)

{

char arg[] = {k,127}; a(17,arg);

byte possibilities = 0; // sets the number of possible moves

// if (x1 > 8 || x2 > 8 || y1 > 8 || y2 > 8) return 0; // ?, @, 9, : are not valid columns

if (CB[x2][y2] > 6 - 6 \* turn && CB[x2][y2] < 13 - 6 \* turn) {

slain = true;

} // checks if (x2,y2) contains an enemy piece

else {

slain = false;

} // no enemy piece

if (CB[x2][y2] > 0 + 6 \* turn && CB[x2][y2] < 7 + 6 \* turn) {

feedbacklight(17,2); // flash red light

g--; return 0; // don't allow zet() to run

} // checks if (x2,y2) contains a friendly piece

if (type == 1) {

possibilities = checkpawn(17,x2,y2,turn);

} // pawn

if (type == 2) {

knight = 1;

possibilities = checkknight(17,x2,y2,turn);

} // knight

else {

knight = 0;

} // no knight

if (type == 3) {

possibilities = checkbishop(17,x2,y2,turn);

} // bishop

if (type == 4 ) {

possibilities = checkrook(17,x2,y2,turn);

} // rook

if (type == 5) {

possibilities = checkqueen(17,x2,y2,turn);

} // queen

if (type == 6) {

possibilities = checkking(17,x2,y2);

} // king

if (possibilities == 0) {

feedbacklight(17,2); // turn red

g--; return 0;

} // no such piece can move to this field

if (possibilities == 1) {

byte t1 = CB[x1][y1]; byte t2 = CB[x2][y2]; // store the values of the types

CB[x2][y2] = CB[x1][y1]; CB[x1][y1] = 0;

if (check(17)) {

CB[x1][y1] = t1; CB[x2][y2] = t2; // put back the original values

feedbacklight(17,2);

g--; return 0;

} // can't do move, in check.

else {

CB[x1][y1] = t1; CB[x2][y2] = t2; // put back the original values

feedbacklight(17,0); //turn off

possibilities = 0;

g--; return 1;

}

} // there is one piece that can move to this field

if (possibilities > 1) {

feedbacklight(17,3); // turn blue

flow = 3; // ask for more information (x1 and y1)

possibilities = 0;

g--; return 1;

} // for example "pawn to E5 has two possibilities

g--;

} // 17

byte print(byte k, boolean d, char b, char c) // d can be true (regular move) or false (wondering)

{

char arg[] = {k,char(d),b,c,127}; a(18,arg);

/\* String piece; byte stuk;

if (d) stuk = CB[x1][y1]; // for ex. bishop on beginning position

else stuk = CB[b][c]; // wondering

if (stuk == 1 || stuk == 7) piece = "pawn";

if (stuk == 2 || stuk == 8) piece = "knight";

if (stuk == 3 || stuk == 9) piece = "bishop";

if (stuk == 4 || stuk == 10) piece = "rook";

if (stuk == 5 || stuk == 11) piece = "queen";

if (stuk == 6 || stuk == 12) piece = "king";

if (d) {

Serial.print("\t\t\t\t\t");

if(turn) Serial.print("Black: ");

else Serial.print("White: ");

Serial.print(piece);

Serial.print(" to ");

Serial.print(char((11 - c) + 62));

Serial.println(char(b+48));

}

else {

Serial.print("\t\t\t\t\t");

if (stuk == 0) Serial.print("There is nothing");

else {

Serial.print("There is a ");

if(stuk > 6) Serial.print("black ");

else Serial.print("white ");

Serial.print(piece);

}

Serial.print(" at ");

Serial.print(char((11 - c) + 62)); // invert board and translate to ascii (therefore turn into char)

Serial.println(char(b+48)); // turn into ascii

}

g--;

\*/

} // 18

byte reset(byte k)

{

char arg[] = {k,127}; a(19,arg);

magnet(19,LOW);

digitalWrite(dirpinx, HIGH); // move motor in negative x direction

digitalWrite(dirpiny, LOW); // move motor in neative y direction

resetx = digitalRead(7);

resety = digitalRead(8);

while (resetx + resety > 0) {

resetx = digitalRead(7);

resety = digitalRead(8);

if (resetx == 1) {

digitalWrite(steppinx, LOW);

digitalWrite(steppinx, HIGH); //This change from LOW to HIGH is necessary for the execution of the stepper.

delayMicroseconds(Speed); //

} // if x button is unpressed

if (resety == 1) {

digitalWrite(steppiny, LOW);

digitalWrite(steppiny, HIGH);

delayMicroseconds(Speed);

}

} //as long as at least one button is unpressed (it turns 0 when pressed)

x3 = 1; // x coordinate of magnet is 1 (1)

y3 = 1; // y coordinate of magnet is 1 (H)

nextxreset = resetnumber;

nextyreset = resetnumber;

magnet(19,HIGH);

feedbacklight(19,0);

g--;

} // 19

byte switchturn(byte k)

{

char arg[] = {k,127}; a(20,arg);

digitalWrite(turn + 12, LOW); // change of side colour

turn = 1 - turn;

digitalWrite(turn + 12, HIGH);

feedbacklight(20,0);

enpassant[turn] = 0; // if it is white's turn, white's last double move will be erased

g--;

} // 20

byte trash(byte k, byte b, byte c) // fuction that deals with capturing and putting the slain piece away (b,c) are the coordinates of the slain piece

{

char arg[] = {k,b,c,127}; a(21,arg);

// (slainpos1, slainpos2) = graveyard white piece, (slainpos2, slainpos1) is graveyard black piece

magnet(21,LOW);

movem(21,(b - x3) \* 2, (c - y3) \* 2); // move to slain piece

x3 = b; y3 = c; // update coordinates

magnet(21,HIGH);

if (turn) {

movegrid(21,slainpos1[totalslainw],slainpos2[totalslainw]);

x3 = slainpos1[totalslainw]; y3 = slainpos2[totalslainw]; // update coordinates

totalslainw++;

} // if turn = 1 (black's turn), then a white piece has been slain, move to (slainpos1[totalslainw],slainpos2[totalslainw])

else {

movegrid(21,slainpos2[totalslainb],slainpos1[totalslainb]);

x3 = slainpos2[totalslainb]; y3 = slainpos1[totalslainb]; // update coordinates

totalslainb++;

} // if turn = 0 (white's turn, then a black piece has been slain

g--;

slain = false;

} // 21

byte xreset(byte k)

{

char arg[] = {k,127}; a(22,arg);

digitalWrite(dirpinx, HIGH);

do {

resetx = digitalRead(7);

digitalWrite(steppinx, LOW);

digitalWrite(steppinx, HIGH); //This change from LOW to HIGH is necessary for the execution of the stepper.

delayMicroseconds(Speed);

} while (resetx > 0); //as long as at least one button is unpressed (it turns 0 when pressed)

resetxmove = LOW;

nextxreset = resetnumber;

x3 = 1;

g--;

} // 22

byte yreset(byte k)

{

char arg[] = {k,127}; a(23,arg);

digitalWrite(dirpiny, LOW);

do {

resety = digitalRead(8);

digitalWrite(steppiny, LOW);

digitalWrite(steppiny, HIGH); //This change from LOW to HIGH is necessary for the execution of the stepper.

delayMicroseconds(Speed); //

} while (resety > 0); //as long as y button is unpressed (it turns 0 when pressed)

resetymove = LOW;

nextyreset = resetnumber;

y3 = 1;

g--;

} // 23

byte zet(byte k)

{

char arg[] = {k,127}; a(24,arg);

magnet(24,LOW);

if (knight == 1) {

xrest = 1; yrest = 1;

autoreset(24,1); // if move from 1-rank/H-file

movem(24,(x1 - x3) \* 2, (y1 - y3) \* 2); //move to beginning field

x3 = x1; y3 = y1; // update coordinates

magnet(24,HIGH);

movegrid(24,x2,y2); // move to second coordinate

flow = 0;

} //movement knight

else {

if (slain == 0 || (abs(x1 - x2) < 2 && abs(y1 - y2) < 2 || type == 7)) {

xrest = x1; yrest = y1;

autoreset(24,1); // if to rank 1 or file H, hit the resetbutton

} // no capturerun (no capture or close capture), or control z

movem(24,(xrest - x3) \* 2, (yrest - y3) \* 2); // move to beginning position

if (slain) {

x3 = xrest; y3 = yrest; // update coordinates

}

else {

x3 = x1; y3 = y1; // update coordiantes

} // not slain

magnet(24,HIGH); // move magnet up

autoreset(24,2); // if to rank 1 or file H, hit the resetbutton

movem(24,(x2 - x3) \* 2, (y2 - y3) \* 2); // move to the final position

x3 = x2; y3 = y2; // update coordinates

flow = 0; // set flow

} //movement non-knight

if(enpassant[turn] == 9) {

trash(24,5 - turn, enpassant[1 - turn]); // remove pawn physically

if (turn) CB[slainpos1[totalslainw - 1]][slainpos2[totalslainw - 1]] = 7 - 6\*turn; // in white's turn a black piece has been slain

else CB[slainpos2[totalslainb - 1]][slainpos1[totalslainb - 1]] = 7 - 6\*turn; // update matrix grave square

CB[5 - turn][enpassant[1 - turn]] = 0; // remove pawn electronically

} // en passant

if(type != 10) {

switchturn(24);

} // if not super user, switchturn

if (nextxreset <= 0 || nextyreset <= 0) {

reset(24);

} // time to reset!

g--;

} // 24