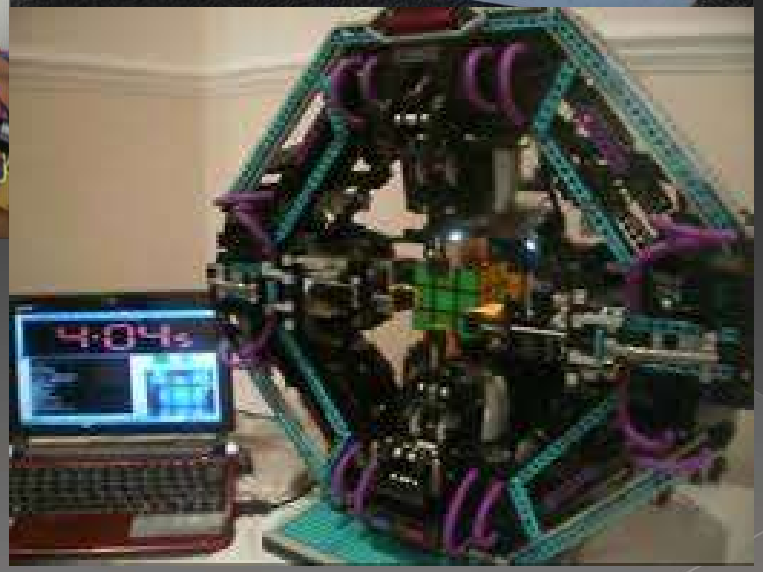
Rubiks-cube robot

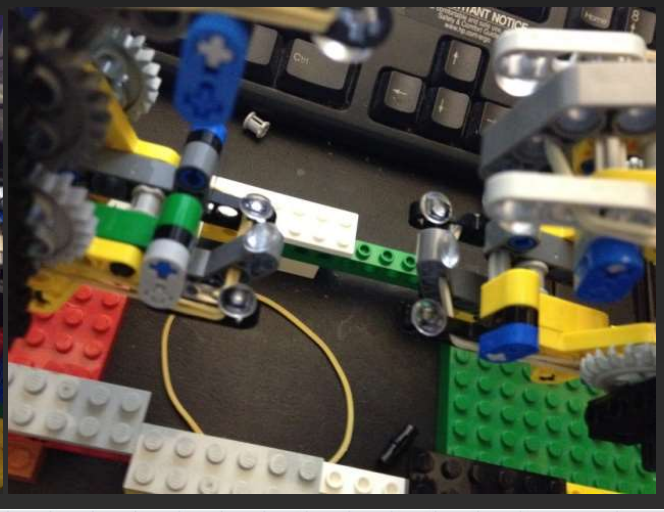
Adrian Stoop
01.07.2016

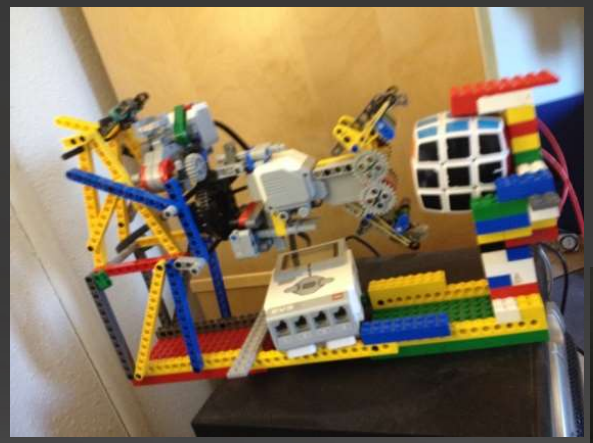
Table of Contents

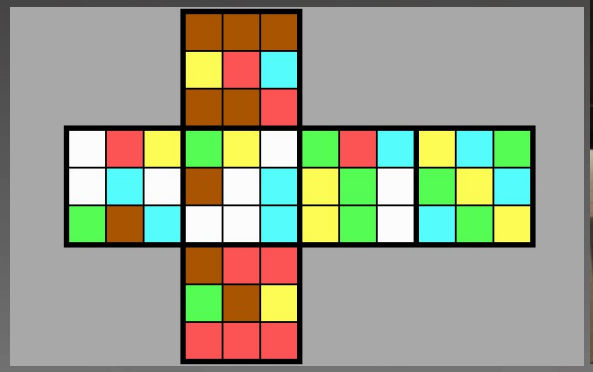
Idea



Proceed







Projektplan

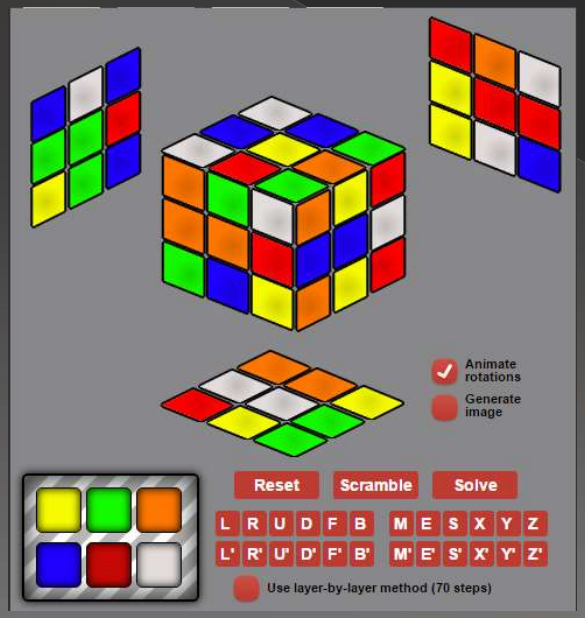
Aufgaben	04.03.2016	11.03.2016	18.03.2016	25.03.2016	01.04.2016	08.04.2016	15.04.2016	22.04.2016	29.04.2016	06.05.2016	13.05.2016	20.05.2016	27.05.2016	03.06.2016	10.06.2016	17.06.2016		
Requirement schreiben	x	x																
Projektplan					x													
Einkauf der Bauteie			x	x														
Erster Roboterprototyp						x	x											
Code für ansteuerung der Motoren								x										
evt Prototyp verbessern								x										
Code für Farbensensor/tests									x									
Roboter komplett zusammenbauen									x	x								
Code für x and o's algorithmus						x	x				x	x	x	x				
Vortrag																		
Projektdoku					x			x		x		x		x		x		
Produktdoku					x			x		x		x		x		x		
Optimierung Code(vgeschwindigkeit)														x	x	x		
Gliederung Texte überarbeiten															x	x		
Soll Stunden(ohne Schulzeit):	0.25	0.25	2	0.5	5	20	20	20	20	25	20	30	20	20	25	20	248	**Total**
Ist Stunden(ohne Schulzeit):	0.5	0	3	1	4.5	25	20	23	21	30	25	24	25	30	35	40	307	**Erledigt**
Differenz:	0.25	-0.3	1	0.5	-0.5	5	0	3	1	5	5	-6	5	10	10	20	59	**Zu tun**

Proceed

- 2nd robot
- Color sensor

Good / bad

- expiration of the code
- Prototype
- Time
- Guide
- Internet



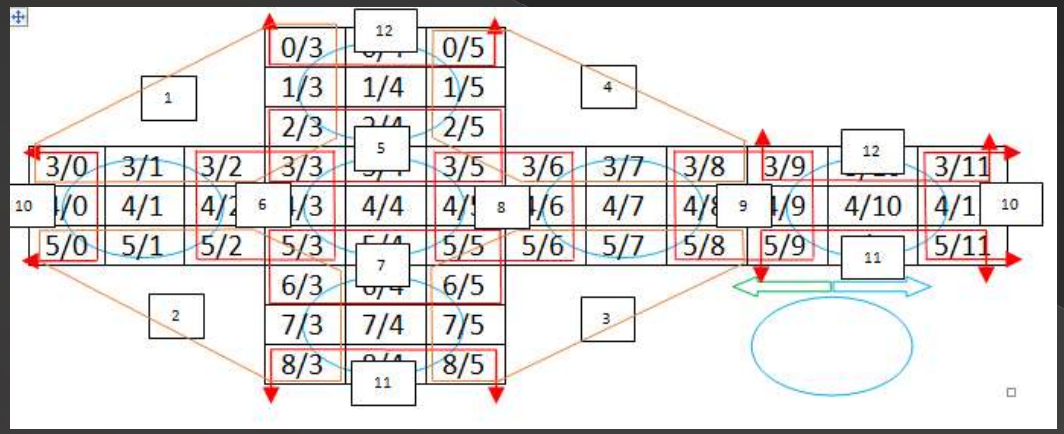
- Connection
- Color sensor
- Debugging
- Trial account

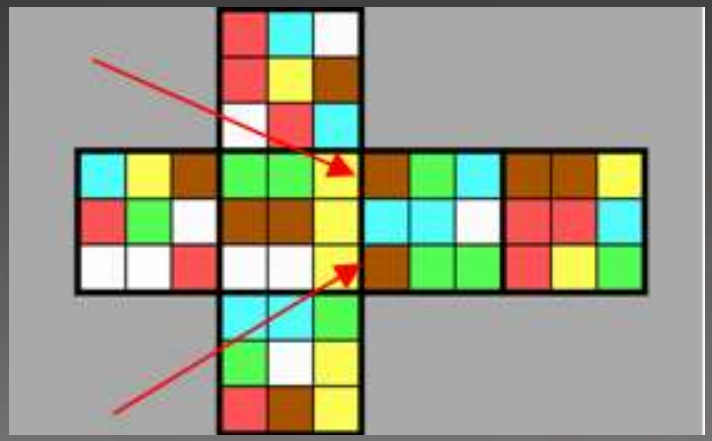
Problems

- Can‘t connect two bricks
- Color sensor are inaccurate
- 2nd robot
- Time
- Wlan-module
- debugging

```
#define T()      R_fehler(0,"Test","#makro");
#define E(a)     int32_disable_gm();printformat("%d",a);W(1);
#define M()      R_Show_Muster();W(1);
```

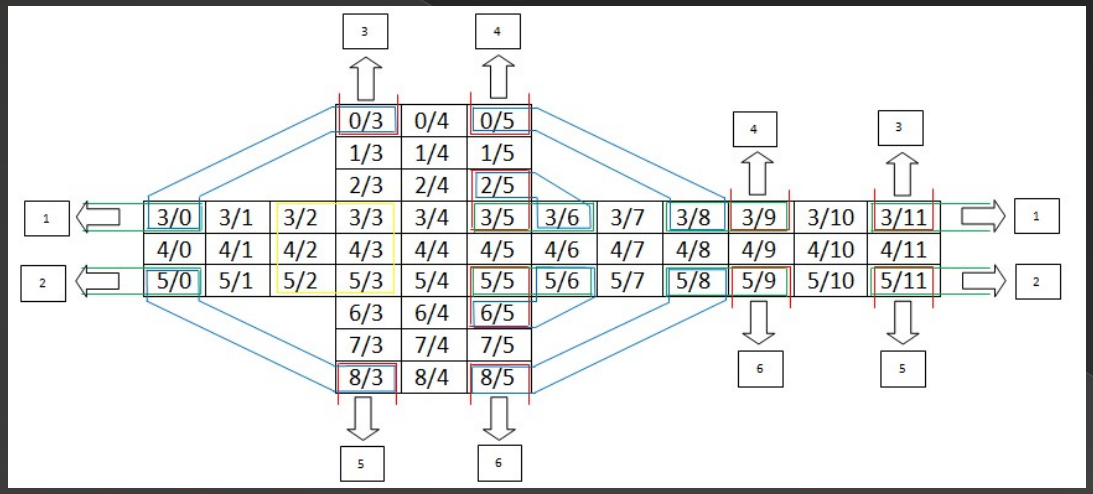
Code

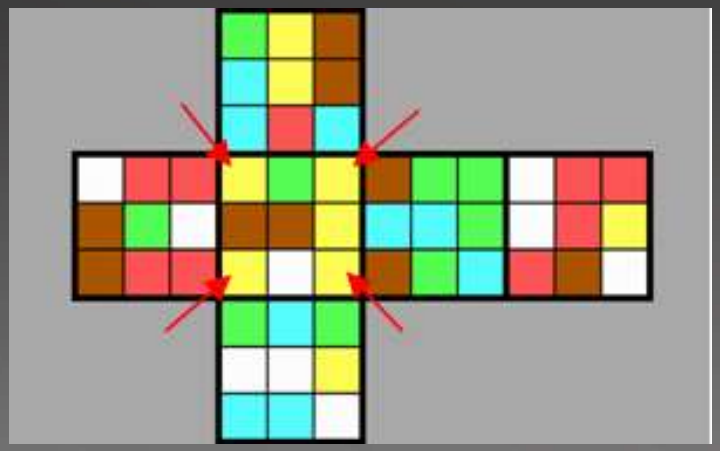




```
//12. Möglichkeit
R_Berechnungen = R_Berechnungen + 2;
if((RubiksFeld[3][9] == RubiksFeld[3][11]) && (RubiksFeld[0][3] == RubiksFeld[0][5]))
    {
        //zweierpaar gefunden, stage beenden.
        R_Stage_0_Status = 1;
        R_Stage_0_2erPos[11] = 1;
    }

//kontrolle ob ein zweierpaar gefunden wurde sonst mache eine drehung und teste alles von vorne
F(var_i,12)
    {
        //ist R_Stage_0_2erStatus > 0, dann ist min. 1 zweiepaar gefunden
        R_Stage_0_2erStatus = R_Stage_0_2erStatus + R_Stage_0_2erPos[var_i];
    }

//wurde min. 1 zweierpaar gefunden?
if(R_Stage_0_2erStatus > 0)
    {
        //ja, verlasse stage 0
        return;
    }
else
    {
        //nein, bewege würfel und teste noch mal(12 seiten möglich, clk und !clk auf 6 seiten)
        //wenn eine andere seitenicht geht wieder in uhrstellung gehen und 90 grad daneben drehen
        //bis es irgendwann klappt

        switch(R_Stage_0_ausrichtung)
            {
                case(0)://F
                    {
                        //standarstellung, wechsle in aussrichtung 1
                        R_Stage_0_ausrichtung = 1;
                        //bewege F und speichere zug im lösungsweg.
                        R_MoveCube_F();
                        R_LWeg[R_LWeg_pointer] = C_F;
                        R_LWeg_pointer++;
                        break;
                        //gehe aus stage 0 raus und geginne den stage 0 von neu(neue ausrichtung)
                    }
                case(1)://Fi
                    {
                        //ausrichtung 1 hatte kein 2er paar gefunden
                        // zurück zur Uhrstellung
                        R_Stage_0_ausrichtung = 0;
                        R_MoveCube_Fi();
                        R_LWeg[R_LWeg_pointer] = 0; //0 ist kein zug, daher nix
                        R_LWeg_pointer--;
                        R_Zuege--;
```

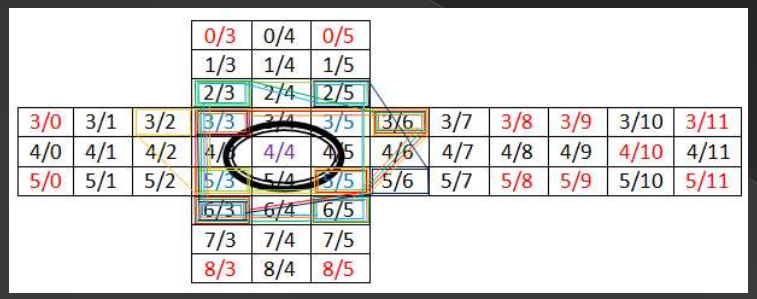
Code

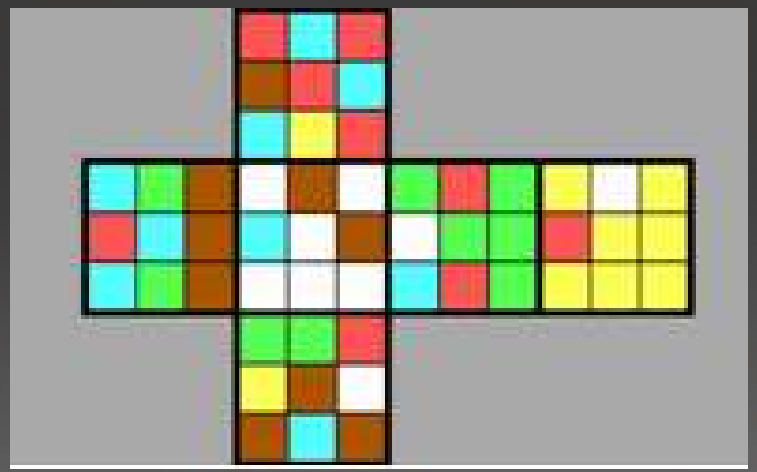




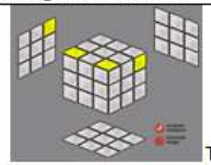
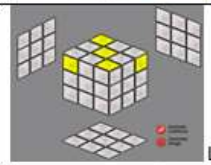
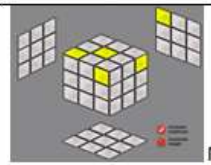
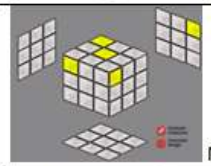
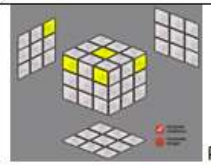
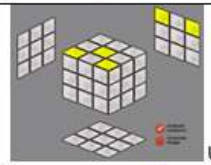
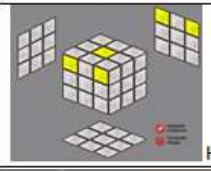
```
//6. blaue möglichkeit A [3/0 0/3]
R_Berechnungen = R_Berechnungen + 2;
if((RubiksFeld[2][3] == RubiksFeld[3][0]) && (RubiksFeld[3][3] == RubiksFeld[0][3]))
    {
        //erster Stein gefunden, drehe ihn an seine Position mit richtiger Ausrichtung.
        flag_stein_1 = 1;
        //drehe stein an seine position
        R_MoveCube_D();
        R_LWeg[R_LWeg_pointer] = C_D;
        R_LWeg_pointer++;
        R_MoveCube_Bi();
        R_LWeg[R_LWeg_pointer] = C_Bi;
        R_LWeg_pointer++;
        R_MoveCube_D();
        R_LWeg[R_LWeg_pointer] = C_D;
        R_LWeg_pointer++;
        R_MoveCube_D();
        R_LWeg[R_LWeg_pointer] = C_D;
        R_LWeg_pointer++;
        R_MoveCube_B();
        R_LWeg[R_LWeg_pointer] = C_B;
        R_LWeg_pointer++;
    }
//6. blaue möglichkeit B [0/3 3/0]
R_Berechnungen = R_Berechnungen + 2;
if((RubiksFeld[2][3] == RubiksFeld[0][3]) && (RubiksFeld[3][3] == RubiksFeld[3][0]))
    {
        //erster Stein gefunden, drehe ihn an seine Position mit richtiger Ausrichtung.
        flag_stein_1 = 1;
        //diese möglichkeit ist nicht möglich
        R_fehler(2,"W[flag_stein_1]","6. blaue möglichkeit B [0/3 3/0]");

    }
```

Code

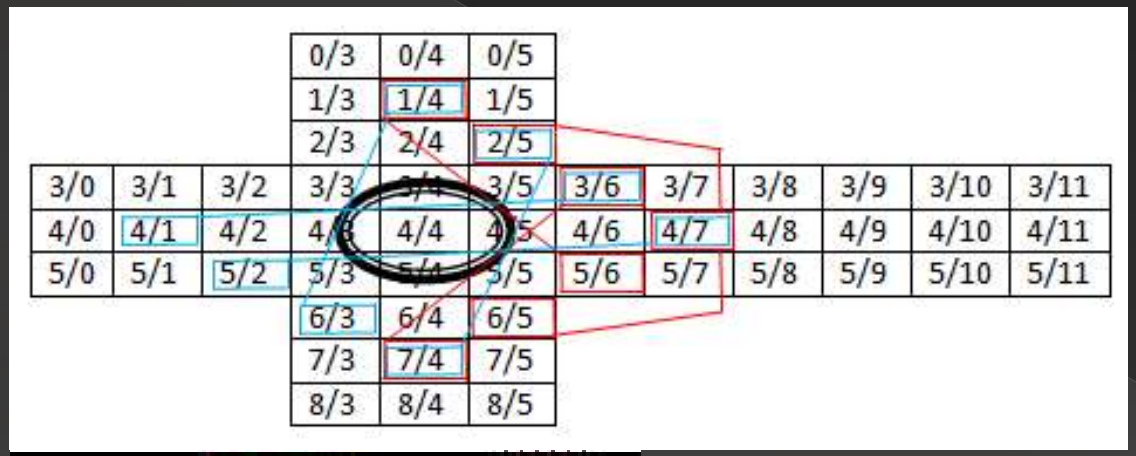


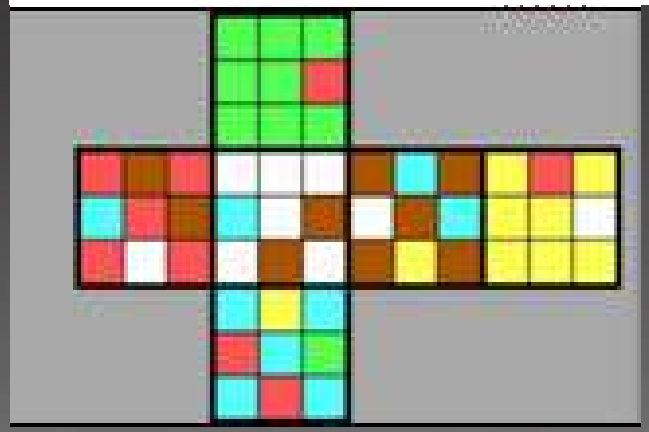


```
//1. möglichkeit von 7
//abfrage Zustand 1 auf allen 4 seiten
F(var_j,4)
    {

        R_Berechnungen = R_Berechnungen + 4;
        if((RubiksFeld[4][4] == RubiksFeld[3][2]) && (RubiksFeld[4][4] == RubiksFeld[3][6
            {
                //benutzte alghoyrtmus

                R_MoveCube_R();
                R_LWeg[R_LWeg_pointer] = C_R;
                R_LWeg_pointer++;
                R_MoveCube_U();
                R_LWeg[R_LWeg_pointer] = C_U;
                R_LWeg_pointer++;
                R_MoveCube_Ri();
                R_LWeg[R_LWeg_pointer] = C_Ri;
                R_LWeg_pointer++;
                R_MoveCube_Ui();
                R_LWeg[R_LWeg_pointer] = C_Ui;
                R_LWeg_pointer++;
                R_MoveCube_Fi();
                R_LWeg[R_LWeg_pointer] = C_Fi;
                R_LWeg_pointer++;
                R_MoveCube_Ui();
                R_LWeg[R_LWeg_pointer] = C_Ui;
                R_LWeg_pointer++;
                R_MoveCube_F();
                R_LWeg[R_LWeg_pointer] = C_F;
                R_LWeg_pointer++;

                //drehe lösungweg in Urfrm und beende Stage *break(); fehlt absichtlich :
                switch(var_j)
                    {
                        case (3):
                            {
                                R_Turn_Cubeandway(VC_nC);
                            }
                        case (2):
                            {
                                R_Turn_Cubeandway(VC_nC);
                            }
                        case (1):
                            {
                                R_Turn_Cubeandway(VC_nC);
                            }
                        //beende stage
                        default:{R_Stage_2_Status = 1; break;}
                    }
                //setzt var_j auf 4 damit man aus der shleiffe fliegt
                var_j = 4;
            }
        else
            {
                //drehe würfel für nächste seite. alle 4 falsch, so ist der würfel wieder
                R_Turn_Cubeandway(VC_C);
            }

    }
```

Möglicher Zustand:	Massnahme(Algorithmus):
 T-Zustand	R U Ri Ui Fi Ui F
 L-Zustand	F Ri Fi Ui Ri U R
 M-Zustand	R U Ri U R 2U Ri
 Mi-Zustand	Ri Ui R Ui Ri 2U R
 Pi-Zustand	B U2 B2 Ui B2 Ui B2 U2 B
 U-Zustand	Ri Fi Ui F U R
 H-Zustand	2R 2U Ri 2U 2R

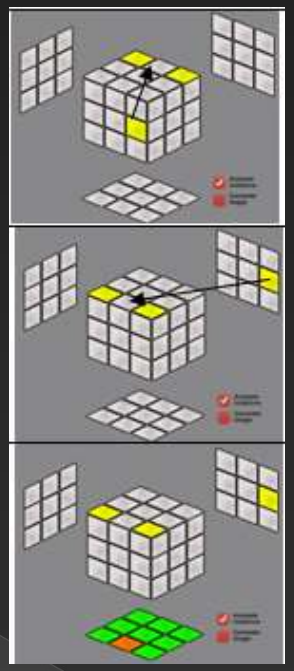
Code

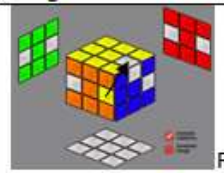
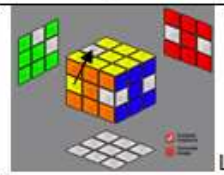




```
//teste eine Seite(4 möglichkeiten)
F(var_j,4)
    {
        R_Berechnungen = R_Berechnungen + 2;
        if((RubiksFeld[6][5] == RubiksFeld[2][5]) && (RubiksFeld[5][6] != RubiksFeld[3][(
            {
                //benutzte alghoyrtmus
                |


                //drehe lösungweg in Urfrm und beende Stage *break(); fehlt absichtlich
                switch(var_j)
                    {
                        case (3):
                            {
                                R_Turn_Cubeandway(VC_nC);
                            }
                        case (2):
                            {
                                R_Turn_Cubeandway(VC_nC);
                            }
                        case (1):
                            {
                                R_Turn_Cubeandway(VC_nC);
                            }
                        //beende stage
                        default:{R_Stage_3_Status = 1; break;}
                    }
                //setzt var_j auf 4 damit man aus der shleiffe fliegt
                var_j = 4;
            }
        else
            {
                //drehe würfel für nächste seite. alle 4 falsch, so ist der würfel wiede
                R_Turn_Cubeandway(VC_C);
            }
    }
```

Code

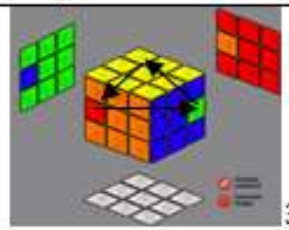
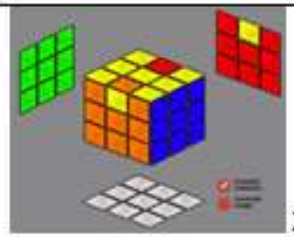


Möglicher Zustand:	Massnahme(Algorithmus):
 R	R E Ri Ei Ri E R
 L	Li Ei L E L Ei Li


```
if((RubiksFeld[3][3] == RubiksFeld[4][6]) && (RubiksFeld[2][3] == RubiksFeld[4][5]))//  4/6 4/5
    {
        //drehe stein an die richtige Position
        R_MoveCube_Ri();
        R_LWeg[R_LWeg_pointer] = C_Ri;
        R_LWeg_pointer++;
        R_MoveCube_Ei();
        R_LWeg[R_LWeg_pointer] = C_Ei;
        R_LWeg_pointer++;
        R_MoveCube_R();
        R_LWeg[R_LWeg_pointer] = C_R;
        R_LWeg_pointer++;
        R_MoveCube_B();
        R_LWeg[R_LWeg_pointer] = C_B;
        R_LWeg_pointer++;
        R_MoveCube_Ei();
        R_LWeg[R_LWeg_pointer] = C_Ei;
        R_LWeg_pointer++;
        R_MoveCube_Bi();
        R_LWeg[R_LWeg_pointer] = C_Bi;
        R_LWeg_pointer++;
    }
```

Code

```
//ist jeder stein falsch?
var_k = 0;
R_Berechnungen = R_Berechnungen + 4;
if((RubiksFeld[7][4] == RubiksFeld[7][5]) && (RubiksFeld[4][7] == RubiksFeld[5][7]) && (var_k == 0)){var_k++;}
if((RubiksFeld[7][4] == RubiksFeld[5][7]) && (RubiksFeld[4][7] == RubiksFeld[7][5]) && (var_k == 0)){var_k++;}
R_Berechnungen = R_Berechnungen + 4;
if((RubiksFeld[4][1] == RubiksFeld[5][1]) && (RubiksFeld[7][4] == RubiksFeld[7][3]) && (var_k == 0)){var_k++;}
if((RubiksFeld[4][1] == RubiksFeld[7][3]) && (RubiksFeld[7][4] == RubiksFeld[5][1]) && (var_k == 0)){var_k++;}
R_Berechnungen = R_Berechnungen + 4;
if((RubiksFeld[4][1] == RubiksFeld[3][1]) && (RubiksFeld[1][4] == RubiksFeld[1][3]) && (var_k == 0)){var_k++;}
if((RubiksFeld[4][1] == RubiksFeld[1][3]) && (RubiksFeld[1][4] == RubiksFeld[3][1]) && (var_k == 0)){var_k++;}
R_Berechnungen = R_Berechnungen + 4;
if((RubiksFeld[1][4] == RubiksFeld[1][5]) && (RubiksFeld[4][7] == RubiksFeld[3][7]) && (var_k == 0)){var_k++;}
if((RubiksFeld[1][4] == RubiksFeld[3][7]) && (RubiksFeld[4][7] == RubiksFeld[1][5]) && (var_k == 0)){var_k++;}
if(var_k == 0)
    {
        R_MoveCube_Ei();
        R_LWeg[R_LWeg_pointer] = C_Ei;
        R_LWeg_pointer++;
        R_MoveCube_B();
        R_LWeg[R_LWeg_pointer] = C_B;
        R_LWeg_pointer++;
        R_MoveCube_B();
        R_LWeg[R_LWeg_pointer] = C_B;
        R_LWeg_pointer++;
        R_MoveCube_E();
        R_LWeg[R_LWeg_pointer] = C_E;
        R_LWeg_pointer++;
        R_MoveCube_B();
        R_LWeg[R_LWeg_pointer] = C_B;
        R_LWeg_pointer++;
        R_MoveCube_B();
        R_LWeg[R_LWeg_pointer] = C_B;
        R_LWeg_pointer++;
    }
var_k = 0;
```

Möglicher Zustand:	Massnahme(Algorithmus):
 3 Falsch	Ei 2B E 2B
 2 Falsch	Mi Ui Mi Ui Mi 2Ui M Ui M Ui M 2Ui

			0/3	0/4	0/5						
			1/3	1/4	1/5						
			2/3	2/4	2/5						
3/0	3/1	3/2	3/3	3/4	3/5	3/6	3/7	3/8	3/9	3/10	3/11
4/0	4/1	4/2	4/3	4/4	4/5	4/6	4/7	4/8	4/9	4/10	4/11
5/0	5/1	5/2	5/3	5/4	5/5	5/6	5/7	5/8	5/9	5/10	5/11
			6/3	6/4	6/5						
			7/3	7/4	7/5						
			8/3	8/4	8/5						

How i test it

```c
if(R_Stage_3_Status == 0)
    {
        //fehler
        R_fehler(3,"IF[R_Stage_3_Status]","diag. oder seite nicht gefunden");
    }
```

```c
float solvecounter = 0;
int as = 3,
    bs = 13,
    cs = 14,
    ds = 16,
    es = 18,
    fs = 23,
    gs = 9,
    hs = 38,
    is = 34;
        //zufallsgenerator
```

```c
//Arrayfeld mit rubiks farben
char RubiksFeld[9][12] = {//Default stellung/Ziel
                          //Oben
        { Dc, Dc, Dc,     Ro,  Ro,  Ro,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Ro,  Ro,  Ro,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Ro,  Ro,  Ro,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},

        //Links           front             Rechts            Back
        { Bl, Bl, Bl,     Ws,  Ws,  Ws,     Gr,  Gr,  Gr,     Ge,  Ge,  Ge},
        { Bl, Bl, Bl,     Ws,  Ws,  Ws,     Gr,  Gr,  Gr,     Ge,  Ge,  Ge},
        { Bl, Bl, Bl,     Ws,  Ws,  Ws,     Gr,  Gr,  Gr,     Ge,  Ge,  Ge},

                          //Unten
        { Dc, Dc, Dc,     Or,  Or,  Or,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Or,  Or,  Or,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Or,  Or,  Or,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
};

char RubiksFeld_ungeloest[9][12] = {//Default stellung/Ziel
                          //Oben
        { Dc, Dc, Dc,     Ro,  Ro,  Ro,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Ro,  Ro,  Ro,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Ro,  Ro,  Ro,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},

        //Links           front             Rechts            Back
        { Bl, Bl, Bl,     Ws,  Ws,  Ws,     Gr,  Gr,  Gr,     Ge,  Ge,  Ge},
        { Bl, Bl, Bl,     Ws,  Ws,  Ws,     Gr,  Gr,  Gr,     Ge,  Ge,  Ge},
        { Bl, Bl, Bl,     Ws,  Ws,  Ws,     Gr,  Gr,  Gr,     Ge,  Ge,  Ge},

                          //Unten
        { Dc, Dc, Dc,     Or,  Or,  Or,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Or,  Or,  Or,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
        { Dc, Dc, Dc,     Or,  Or,  Or,     Dc,  Dc,  Dc,     Dc,  Dc,  Dc},
};
```

```c
void R_Scan(void)
    {


        char wahl = 0;
        char R_Scan_counter = 0;

        while(R_Scan_counter != 40)
            {

                R_Scan_counter++;

                wahl = ((as + bs + cs + ds + es + fs) * (gs + hs + is))% 16 + 1;

                cs = (as + bs) % 13;
                fs = (ds + es) % 17;
                is = (gs + hs) % 19;

                as = as % 20;
                bs = bs % 20;
                ds = ds % 20;
                es = es % 20;
                gs = gs % 20;
                hs = hs % 20;

                switch(wahl)
                    {
                        case(0):
                            {
                                as = as + 4;
                                bs = bs + 2;
                                ds = ds + 3;
                                es = es + 5;
                                gs = gs + 2;
                                hs = hs + 4;
                                R_MoveCube_R();
                                break;
                            }
                        case(1):
                            {
                                as = as + 2;
                                bs = bs + 6;
                                ds = ds + 5;
                                es = es + 1;
                                gs = gs + 3;
                                hs = hs + 4;
                                R_MoveCube_Ri();
                                break;
                            }
                        case(2):
                            {
                                as = as + 8;
```

```c
else
    {
        R_Berechnungen = R_Berechnungen + 4;
        //hinten
        if((RubiksFeld[7][4] == RubiksFeld[2][3]) && (RubiksFeld[4][1] == R
            {
                //oben um 180' drehen
                R_MoveCube_U();
                R_LWeg[R_LWeg_pointer] = C_U;
                R_LWeg_pointer++;
                R_MoveCube_U();
                R_LWeg[R_LWeg_pointer] = C_U;
                R_LWeg_pointer++;
            }
        else
            {
                //fehler
                R_fehler(3,"IF[seite richtig ausgedreht]","last else...");
            }
    }
```

```c
void R_Paint(int x,int y,char feld)
    {
        switch(feld)
            {
                case(Ws):{int32_rechteck_0x13(x,y,x+19,y+19,WHITE);break;}
                case(Ro):{int32_rechteck_0x13(x,y,x+19,y+19,RED);break;}
                case(Bl):{int32_rechteck_0x13(x,y,x+19,y+19,BLUE);break;}
                case(Ge):{int32_rechteck_0x13(x,y,x+19,y+19,YELLOW);break;}
                case(Gr):{int32_rechteck_0x13(x,y,x+19,y+19,GREEN);break;}
                case(Or):{int32_rechteck_0x13(x,y,x+19,y+19,BROWN);break;}
                default: {R_fehler(9,"R_Paint","Feldinhalt nicht korrekt!");break;}
            }
    }
```

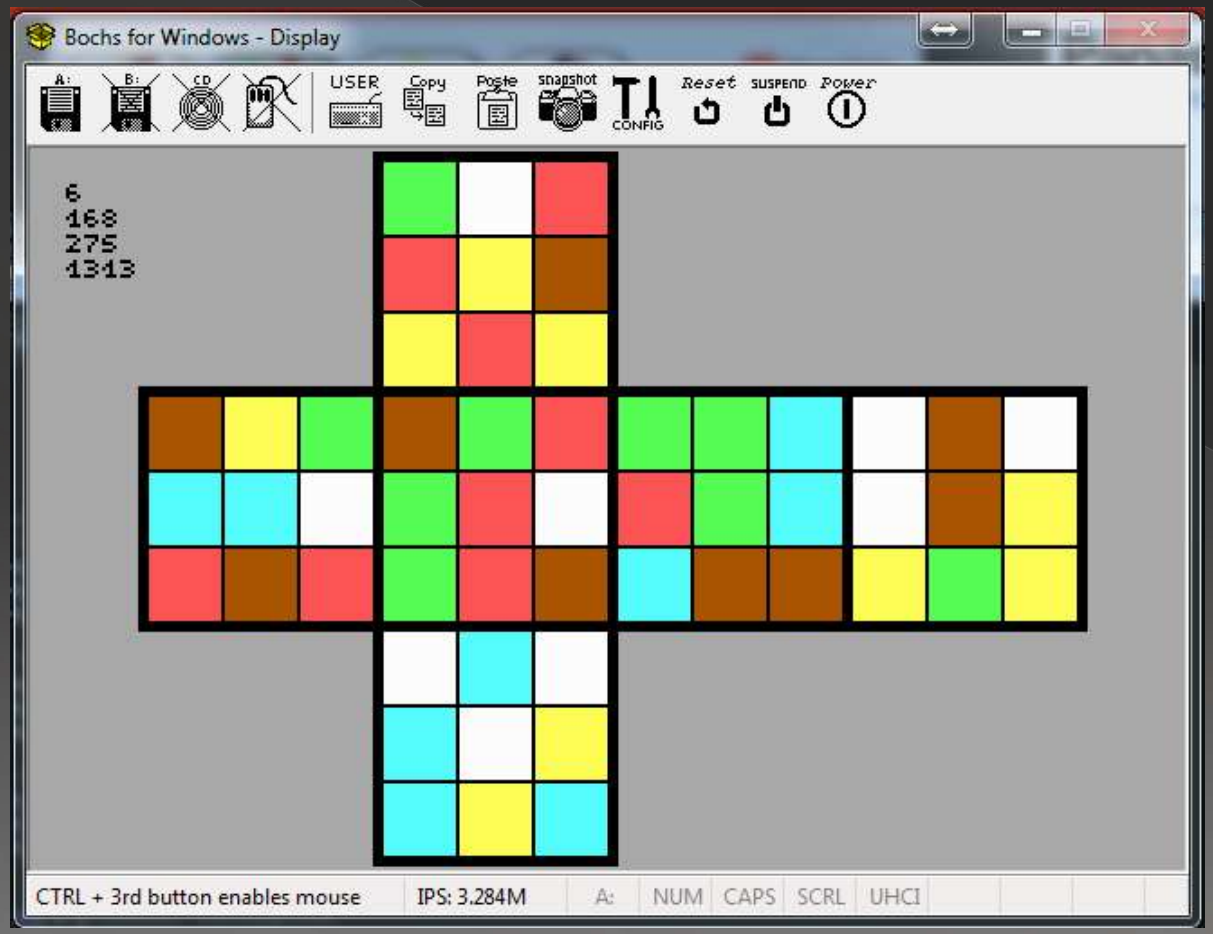
Code

```
//was muss gedreht werdne
switch(R_LWegReal[pointer])
    {
        case(C_R):
            {
                cube_setmotor(hinten);
                cube_setmotor(kralle);

                cube_setmotor(hinten);
                cube_setmotor(kralle);

                cube_setmotor(hinten);
                cube_setmotor(kralle);

                cube_setmotor(clockk);

                cube_setmotor(hinten);
                cube_setmotor(kralle);
                pointer++;
                break;
            }
```

```
case(hinten):
    {
        switch(pos)
            {
                case(kralle):
                    {
                        moveMotorTarget(motorA, 160, -50);sleep(1500);
                        cube_oldpos = kralle;
                        break;
                    }
                case(halte):
                    {
                        moveMotorTarget(motorA, 140, -50);sleep(1500);
                        cube_oldpos = halte;
                        break;
                    }
```

```
case(C_Mi):
    {
        R_LWegReal[pointer] = C_Ri;
        pointer++;
        R_LWegReal[pointer] = C_L;
        pointer++;
        break;
    }
case(C_S):
    {
        R_LWegReal[pointer] = C_B;
        pointer++;
        R_LWegReal[pointer] = C_Fi;
        pointer++;
        break;
    }
```

Demonstration




```
//drgeht den zug Fi
void R_MoveCube_Ui(void)
    {
        Temp_1 = RubiksFeld[6][3];
        Temp_2 = RubiksFeld[6][4];
        Temp_3 = RubiksFeld[6][5];

        RubiksFeld[6][3] = RubiksFeld[3][2];
        RubiksFeld[6][4] = RubiksFeld[4][2];
        RubiksFeld[6][5] = RubiksFeld[5][2];

        RubiksFeld[3][2] = RubiksFeld[2][5];
        RubiksFeld[4][2] = RubiksFeld[2][4];
        RubiksFeld[5][2] = RubiksFeld[2][3];

        RubiksFeld[2][5] = RubiksFeld[5][6];
        RubiksFeld[2][4] = RubiksFeld[4][6];
        RubiksFeld[2][3] = RubiksFeld[3][6];

        RubiksFeld[5][6] = Temp_1;
        RubiksFeld[4][6] = Temp_2;
        RubiksFeld[3][6] = Temp_3;

        Temp_1 = RubiksFeld[5][5];
        Temp_2 = RubiksFeld[5][4];

        RubiksFeld[5][5] = RubiksFeld[5][3];
        RubiksFeld[5][4] = RubiksFeld[4][3];

        RubiksFeld[5][3] = RubiksFeld[3][3];
        RubiksFeld[4][3] = RubiksFeld[3][4];

        RubiksFeld[3][3] = RubiksFeld[3][5];
        RubiksFeld[3][4] = RubiksFeld[4][5];

        RubiksFeld[3][5] = Temp_1;
        RubiksFeld[4][5] = Temp_2;

        R_Wechsel = R_Wechsel + 25; R_Zuege++;
    }
```

Question