# Bundeswettbewerb Informatik – Firat Dündar

Zur Bearbeitung der Aufgaben habe ich C# benutzt (Microsoft Visual Studio 2015 Community) in der .NET Framework 2.0.

Hier folgt nun die Dokumentation, in der ich nur die wichtigsten Quellcode-Stücke eingefügt habe –den kompletten Quellcode zu den einzelnen Aufgaben finden Sie in den jeweilligen Ordner.

Mehr Aufgaben habe ich leider aufgrund des Zeitmangels durch die Schule und dadurch, dass ich erst spät vom Wettbewerb gehört habe, nicht mehr geschafft. Dennoch möchte ich die Aufgaben gerne einschicken.

# Aufgabe 1)

Ich habe in dieser Aufgabe das Brute Force Verfahren benutzt. Hier werden alle Kombinationen von Fächern ausprobiert, bis die kleinste Zahl gefunden wird, die größer als 20 ist. Es gibt 2 Tasten in der Anwendung: “Ausgabefach auffüllen” und “Verpacken”. “Ausgabefach auffüllen” sucht nach der kleinsten Zahl und leert die jeweilligen Fächer ins Ausgabefach aus. “Verpacken” verpackt die Ballons, die sich anschließend im Ausgabefach befinden. Außerdem kann man die Füllfolge während der Laufzeit verändern und anschließend die Fächer neu initialisieren, damit diese neue Füllfolge auch angewandt wird.

Es wurde in der Aufgabe nicht explizit gesagt, was passieren soll, wenn alle Zahlen in der Füllfolge bereits benutzt wurden – ich habe dies so programmiert, dass wenn alle Zahlen benutzt wurden, eine zufällige Zahl aus der Füllfolge benutzt wird. Außerdem habe ich einige kleine Hilfsmittel benutzt, wie z.B. eine Log-Funktion, die einen neuen Eintrag in der ListView erstellt, damit das Programm übersichtlicher dargestellt wird.

## Fach(i)-Funktion

private void Fach(int n, bool istStart = false)

{

int startWert = speicherFächer[n];

if (!istStart)

{

// Fach entleeren

ausgabeFach += speicherFächer[n];

label21.Text = "Es befinden sich zur Zeit " + ausgabeFach.ToString() + " Luftballons im Ausgabefach.";

}

// Finde unbenutzte Zahl

bool zahlGefunden = false;

for (int j = 0; j < füllFolge.Length; j++)

{

// Existiert sie bereits?

bool zahlExistiert = false;

for (int mk = 0; mk < speicherFächer.Length; mk++)

{

if (speicherFächer[mk] == int.Parse(füllFolge[j].ToString()))

{

zahlExistiert = true;

break;

}

}

if (!zahlExistiert)

{

// Kann benutzt werden

speicherFächer[n] = int.Parse(füllFolge[j].ToString());

zahlGefunden = true;

break;

}

}

if (!zahlGefunden)

{

// Wähle einen zufälligen Wert

Random rand = new Random();

speicherFächer[n] = int.Parse(füllFolge[rand.Next(0, füllFolge.Length)].ToString());

}

foreach(Control currentControl in groupBox1.Controls)

{

if(currentControl.Name.Equals(("sp" + n).ToString())){

currentControl.Text = speicherFächer[n].ToString();

}

}

if(istStart)

log("Mit " + speicherFächer[n].ToString() + " Ballons aufgefüllt.", n.ToString());

else

log(startWert.ToString() + " Ballons ins Ausgabefach gelegt und das Fach mit " + speicherFächer[n].ToString() + " Ballons aufgefüllt.", n.ToString());

}

## Verpacken() Funktion

private void Verpacken()

{

log("Es wurden " + ausgabeFach.ToString() + " Luftballons verpackt.", "Alle");

ausgabeFach = 0;

label21.Text = "Es befinden sich zur Zeit " + ausgabeFach.ToString() + " Luftballons im Ausgabefach.";

}

## Die Rekursiv-Funktion (Brute Force)...

void rek(ref List<int> benutzteFächer, int aktuell, ref int nW, ref List<int> nWF)

{

for (int i=1; i<speicherFächer.Length; i++)

{

// Überprüfe, ob diese Speicherfach bereits in benutzteFächer ist

bool isBenutzt = benutzteFächer.Contains(i);

if (!isBenutzt)

{

benutzteFächer.Add(i);

int neuerWert = aktuell + speicherFächer[i];

if (neuerWert < 20)

{

rek(ref benutzteFächer, neuerWert, ref nW, ref nWF);

}

else

{

if(neuerWert < nW || nW < 20)

{

// Ist größer als 20 aber kleiner als nW

nWF.Clear();

nWF = new List<int>(benutzteFächer);

nW = neuerWert;

}

}

benutzteFächer.Remove(i);

}

}

}

## ... die wie folgt aufgerufen wird (nach dem Button-Klick)

// Damit die Rekursiv-Funktion später weiß, welche Fächer bereits kombiniert wurden

List<int> benutzteFächer = new List<int>();

int nW = 0; // Hier wird der niedrigste Wert gespeichert

List<int> nWF = new List<int>(); // Kombination aus Fächern, die zusammen “nW” ergeben

rek(ref benutzteFächer, 0, ref nW, ref nWF);

// “ref” bedeuetet, dass die in dieser Funktion initialisierte Variable benutzt werden soll, anstatt dass eine neue Instanz der Variable erstellt wird

# Aufgabe 2)

Bei LAMA gibt es einige Einschränkungen – die minimal zugelassene Rastergröße beträgt 3 x 3 und die maximale beträgt 8 x 8. Standardmäßig startet das Programm im 5 x 5 Modus. Die Varianten können über das Dropdown-Menü in der Toolstrip ausgewählt werden. Zur Auswahl stehen die 1. Variante, wo die LED inkl. die angrenzenden LEDs (links, rechts, oben, unten) umgeschaltet werden und die 2. Variante, wo die LED selbst und das ganze Rechteck oben links von der LED aus umgeschaltet wird.

Eingeschaltete LEDs sind Gelb, ausgeschaltete LEDs sind weiß (Hintergrundfarbe).

NOTIZ: Die LEDs werden benannt, damit diese später unterschieden werden können. Beispiel: die 4. LED in der 1. Reihe hat den Namen “LED4”. Die 3. LED in der 4. Reihe hat den Namen “LED27”. Dies erkennt man an der folgenden Darstellung:

\* 1 2 3 4 5 6 7 8

\* 9 10 11 12 13 14 15 16

\* 17 18 19 20 21 22 23 24

\* 25 26 27 28 29 30 31 32

\* 33 34 35 36 37 38 39 40

\* 41 42 43 44 45 46 47 48

\* 49 50 51 52 53 54 55 56

\* 57 58 59 60 61 62 63 64

## Die LED-Umschalten Funktion

private void LED\_Click(object sender, EventArgs e)

{

// ID abrufen

int ID = Convert.ToInt16(((Control)sender).Name.Replace("LED", string.Empty));

if (varianteToolStripMenuItem1.Checked)

{

// 1. Variante

LED\_Umschalten(ID);

//MessageBox.Show(ID.ToString() + " Mod 8 = " + (ID % 8).ToString());

if(ID - 8 > 0)

{

LED\_Umschalten(ID - 8);

}

if(ID - 1 >= 0 && ID % 8 != 1)

{

LED\_Umschalten(ID - 1);

}

if(ID % 8 + 1 < 8 && ID % 8 != 0)

{

LED\_Umschalten(ID + 1);

}

if(ID+8 < 65)

{

LED\_Umschalten(ID + 8);

}

}else

{

// 2. Variante

// Spalte und Reihe anhand der ID berechnen

int reihe = ID / 8;

if (ID % 8 != 0)

reihe++;

int spalte = ID % 8;

if (spalte == 0)

spalte = 8;

for(int nR = 1; nR<=reihe; nR++)

{

for(int nS = 1; nS<=spalte; nS++)

{

LED\_Umschalten((nR - 1) \* 8 + nS);

}

}

}

}

## Die LEDs werden wie folgt erstellt

private void Form1\_Load(object sender, EventArgs e)

{

Random rand = new Random();

for(int reihe=1; reihe<=8; reihe++)

{

for(int spalte=1; spalte<=8; spalte++)

{

PictureBox nwPbox = new PictureBox();

nwPbox.BorderStyle = BorderStyle.FixedSingle;

nwPbox.Size = new Size(42, 42);

nwPbox.BackColor = Color.White;

nwPbox.Location = new Point((spalte-1) \* 42 + (spalte-1) \* 6, (reihe-1) \* 42 + (reihe-1) \* 6);

nwPbox.Name = "LED" + ((reihe-1) \* 8 + spalte).ToString();

nwPbox.Click += LED\_Click;

// Zufällig aktivieren

if (rand.Next(0, 100) >= 50)

nwPbox.BackColor = Color.Yellow;

panel1.Controls.Add(nwPbox);

}

}

// x = x + 11

// y = x + 66 aufgrund der oberen Leiste (Windows 10), und ToolStrip

this.Size = new Size(251, 317);

// y = x - 16

panel1.Size = new Size(251, 235);

}

# Aufgabe 4)

Hier habe ich einige Zusatzfunktionen eingebaut, wie z.B. eine “Voraussichtliche Dauer”-Anzeige und eine “Qualität reduzieren (um 25%)” Funktion, die funktioniert, welche jedoch eher zu Entwicklungszwecken eingebaut wurde, da das Ergebnis nicht eindeutig ist. Da diese nicht Teil der Aufgabe waren, werde ich diese nicht weiter erläutern. Dennoch sind diese noch im Quellcode zu finden.

Die “if(isClosing) return;” Codezeilen sind dafür da, damit der Code abgebrochen werden kann (FormClosing-Event setzt den Wert auf true). Man könnte auch Thread.abort() aufrufen, jedoch finde ich, dass die von mir verwendete Lösung die bessere ist, da hier der aktuell ausgeführte Code auch zu Ende ausgeführt werden kann.

## Bild scannen Funktion

private void workNow()

{

// Berechne Pixel; ***totalPixels*** und ***procPixels*** werden außerhalb der Funktion deklariert damit der Timer für die Berechnung der Dauer darauf zugreifen kann

totalPixels = nwB.Width \* nwB.Height;

this.Invoke((MethodInvoker)delegate {

progressBar1.Maximum = totalPixels;

});

for (int x=0; x<nwB.Width; x++)

{

if (isClosing)

return;

for(int y=0; y<nwB.Height; y++)

{

if (isClosing)

return;

incrPBar();

// Pixel-Farbe abrufen

Color nwC = Color.FromArgb(nwB.GetPixel(x, y).ToArgb());

Point[] locations = { new Point(x+1, y), new Point(x-1,y), new Point(x, y-1), new Point(x, y+1), new Point(x - 1, y-1), new Point(x + 1, y-1), new Point(x - 1, y+1), new Point(x + 1, y+1) };

foreach (Point curLoc in locations){

if(curLoc.X >= 0 && curLoc.X < nwB.Width && curLoc.Y >= 0 && curLoc.Y < nwB.Height)

{

if (Color.FromArgb(nwB.GetPixel(curLoc.X, curLoc.Y).ToArgb()) == nwC)

{

nwB.SetPixel(curLoc.X, curLoc.Y, Color.White);

nwB.SetPixel(x, y, Color.White);

}

}

}

}

}