Dokumentation BWINF 2021 - Aufgabe 1: „Schiebeparkplatz“

Team-ID: **TODO!!!**

Team-Name: **TODO!!!**

Bearbeiter: Karl R. Jahn

Dresden, der 03. August 2021

Inhalt

1. Lösungsidee

Das Problem ist, den kürzesten Weg ein Auto wegzuschieben zu finden, unter der Beachtung und davorliegen Entfernung der anderen Hindernisse (Autos). Zur Lösung des Problems versuchte ich zuerst n Regeln zu formulieren:

1. Sollte das ausfahrende Auto kein Hindernis vor sich haben, so kann es Ausfahren und nichts muss veranlasst werden.
2. Wird das Auto blockiert so ist zuerst in die eine Richtung (z.B. Rechts) zu sehen und zu prüfen wie weit das Auto in diese Richtung fahren müsste, wie viele Autos ggf. ausweichen müssten, wie weit diese dann Fahren müssten und ob es überhaupt möglich ist, oder ob diese an den Rand fahren würden.
3. Ist dies getan, so wird Regel 2 auf die andere Seite angewandt und die, welche die wenigsten Schritte beansprucht und überhaupt möglich ist wird verwendet. Nun werden die Autos alle einzeln nacheinander von der ermittelten Seite zum blockierten Auto hin „verschoben“.

2. Umsetzung

Zur Umsetzung in Form eines Programmes, nutzte ich C# als Programmiersprache. Als Framework nutzte ich .NET Core in der Version 3.1.

```
if (!File.Exists(this.path)) {
    MessageBox.Show("Datei exsitiert nicht mehr!");
    return;
}

string[] content = new string[0];

StreamReader file = new StreamReader(this.path);
string line = file.ReadLine();

while(line != null)
{
    Array.Resize(ref content, content.Length + 1);
    content[^1] = line;
    line = file.ReadLine();
}

file.Close();


int nCars = (content[0].ToUpper())[2] - (content[0].ToUpper())[0] + 1;
Tuple<char, CarPart>[] parkingSpot = new Tuple<char, CarPart>[nCars];

for (int i = 0; i < nCars; i++)
    parkingSpot[i] = new Tuple<char, CarPart>((char)0, CarPart.Null);

int max = Convert.ToInt32(content[1]);

for (int i = 0; i < max; i++)
{
    int pos = Convert.ToInt32(content[2 + i][2..]);
    char car = (content[2 + i].ToUpper())[0];
    parkingSpot[pos] = new Tuple<char, CarPart>(car, CarPart.X1);       // vorne  -> X1
    parkingSpot[pos + 1] = new Tuple<char, CarPart>(car, CarPart.X2);   // hinten -> X2
}

for (int i = parkingSpot.Length - 1; i >= 0; i--)
{
    if (parkingSpot[i].Item1 == 0)
       Output(((char)(i + 65)) + ": ", false);
    else
    {
        bool direktion = false; // true => left, false => rigth
        bool x1 = parkingSpot[i].Item2 == CarPart.X1;
        bool null_pos_r = false, null_pos_l = false;
        int break_pos = 0;
        int sec_break_pos_r = 0;
        int sec_break_pos_l = 0;

        for (int t = i; t < parkingSpot.Length; t++)
        {
            if (parkingSpot[t].Item1 == 0)
                {
                    if (x1 || null_pos_r) { break_pos = t; break; };
                    null_pos_r = true;
                    sec_break_pos_r = t;
                }
        }

        for (int t = i, x = i; t >= 0; t--, x++)
        {
            if (parkingSpot[t].Item1 == 0)
```

```
            {
                if (!x1 || null_pos_l)
                {
                    if (break_pos == 0 || break_pos > (x1 ? x : x - 1))
                    {
                        break_pos = t;
                        sec_break_pos_r = sec_break_pos_l;
                        direktion = true;
                    }
                    break;
                }
                null_pos_l = true;
                sec_break_pos_l = t;
            }
            else if (break_pos != 0 && x > break_pos + 1) break;
        }

        int add = direktion ? 1 : -1;
        string direktion_s = direktion ? "left" : "rigth";
        string output = ((char)(i + 65)) + ": ";
        Tuple<char, CarPart>[] tempPS = new Tuple<char, CarPart>[parkingSpot.Length];
        parkingSpot.CopyTo(tempPS, 0);
        Tuple<char, CarPart> clear = new Tuple<char, CarPart>((char)0, CarPart.Null);

        for (int t = break_pos; t != i; t += add)
        {
            if (tempPS[t].Item1 == 0)
            {
                if (tempPS[t + add].Item1 == 0)
                {
                    tempPS[t] = tempPS[t + add * 2];
                    tempPS[t + add] = tempPS[t + add * 3];
                    tempPS[t + add * 2] = clear;
                    tempPS[t + add * 3] = clear;
                    output += tempPS[t].Item1 + " 2 to the " + direktion_s + ", ";
                }
                else
                {
                    tempPS[t] = tempPS[t + add];
                    tempPS[t + add] = tempPS[t + add * 2];
                    tempPS[t + add * 2] = clear;
                    output += tempPS[t].Item1 + " 1 to the " + direktion_s + ", ";
                }
            }
        }
        Output(output[..^2], false);
    }
}
```