Dokumentation BWINF 2021 - Aufgabe 1: „Schiebeparkplatz“

Team-ID: 00564

Team-Name: „SRZ info3 Gruppe 1“

Bearbeiter: Karl Jahn

Dresden, der 20. November 2021

Inhalt

1. Lösungsidee

Meine Idee ist für jedes Auto schrittweise zu überprüfen, ob dieses herauskann, kann es, so muss ja eigentlich nichts mehr getan werden. Kann es nicht raus, so beginnt nun die Arbeit für die eigentliche Idee. Und zwar soll in beide Richtungen von dem Parkplatz nach Lücken gesucht werden.

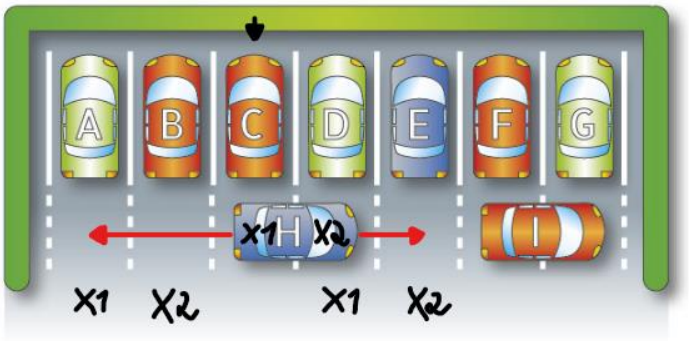

Abb. 1: Vorgehensweise für Auto H

Angenommen wir sind bei Auto C, des Beispiels 0 (das des Aufgabenblattes) dann versperrt X1 von Auto H den Weg (siehe Abb. 1). Also muss Auto H bewegt werden. Hierzu müsste Auto H zwei Lücken nach links fahren, oder aber eine nach vorn. Also nehmen logischer Weise den kürzesten Weg (den nach rechts) und geben ihn aus.

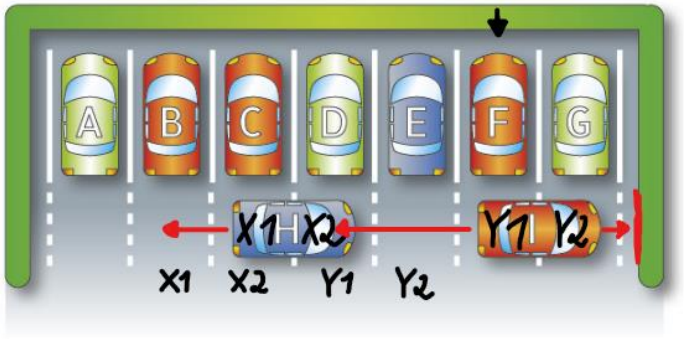

Abb. 2: „Problemauto“ F

Auto F vereinigt viele „Problem“, zum einen kann Auto I nicht nach rechts, da dort das Ende des Parkplatzes ist, zum anderen kann es auch nicht ohne weiteres nach links, da dort Auto H ist.

Meine Idee hierfür ist es, zuerst die zu erledigenden Schritte in beide Richtungen zu zählen, bzw. zu prüfen, ob diese Richtung überhaupt möglich ist und dann die/der beste/möglichen Richtung entgegensetzt die Autos von dem aus Endpunkt in diese Richtung zu verschieben.

2. Umsetzung

2.1 Algorithmus

Ich denke das ganze lässt sich besser verstehen, wenn man das Struktogramm dazu sieht. Auf Abb. 3 ist die „Hauptmethode“ die beim Startklickevent ausgelöst wird vereinfacht zu sehen.

Hinweis: Methoden ließen sich leider nicht normengerecht darstellen, weshalb ich den einen Methodenaufruf mit Klammern in C#-Stiel darstellte.

Die Methode **FindBestDirketion** gibt die beste Richtung aus, -1 für links und 1 für rechts. Ich habe für diese hier kein Struktogramm eingefügt, da es alles andere als Strukturiert aussieht (aufgrund der vielen Verzweigungen) und würde deshalb lieber später auf den Quellcodeteil (in Abschnitt 2.2) verweisen und genauer auf diese eingehen.

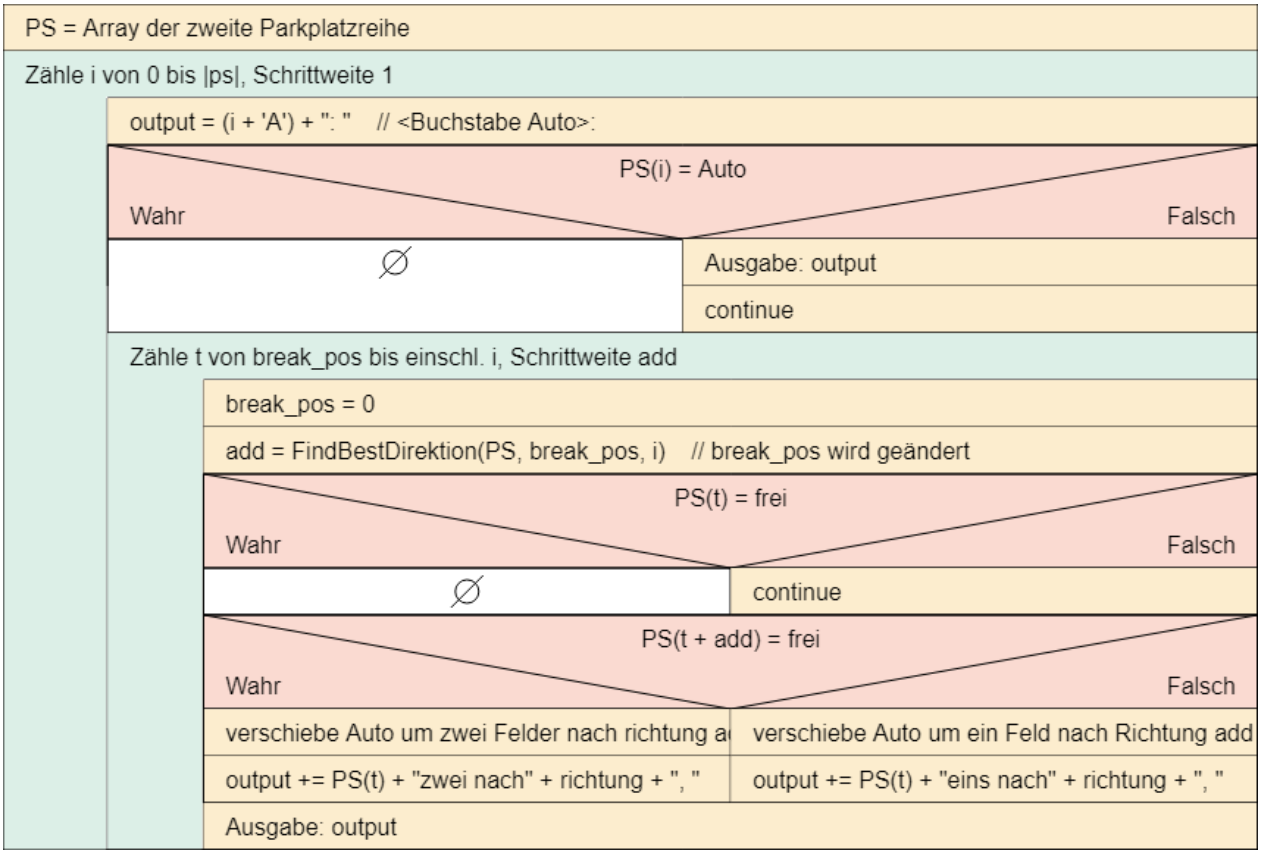

Abb. 3: Struktogramm für die „Hauptmethode“

2.2 Implementation

Zur Umsetzung nutzte ich C# (.NET Core 3.1) als Programmiersprache, da sich hierüber gute UIs entwickeln lassen (Windows Forms) und ich mittels einer UI die Benutzung des Programmes vereinfachen wollte. Die Executable benötigt die beiden JSON - Konfigurationsdateien und die Aufgabe1.dll, welche sich im „bin“ - Ordner mit befinden.

Der Quellcode verteilt sich auf 4 Dateien:

Main.cs	Main.Desinger.cs	Program.cs	Resources.Desinger.cs
Hier befindet sich der wichtigste Teil des Codes mit dem Algorithmus zur Verschiebung der Autos und dem Handling des UI.	Hier werden alle Steuerelemente erzeugt und Initialisiert. Die Datei ist teilw. Autogeneriert.	Hier befindet sich die **Main**-Methode, welche ein Objekt der Klasse Main (in Main.cs) erstellt und die Main-Form zeichnet/aufruft.	Diese Datei ist vollständig Autogeneriert und wird nur zur Kompilierung benötigt. Genauso wie: • „Aufgabe1.csproj“, • „Main.resx“ und • „Resources.resx“.

Wie bereits erwähnt (und der Name auch sagt) sucht **FindBestDirketion** nach der besten Verschieberichtung. Dafür braucht die Methode die querstehende Parkplatzreihe (welche ich überall als 2er-Tuple-Array darstelle) und den die Position des Autos, welches ausfahren soll.

Nun zurück gibt das ganze (logischer Weise) die Richtung, in welche das Auto soll. Diese gebe ich der Einfachheit halber als **signed int** zurück (siehe 2.1 Algorithmus). Dies hat den Grund, dass ich bei der Ausgabe so einfacher in die richtige Richtung iterieren kann.

Weiterhin gibt die Methode den Index der Position an die das erste Auto hingeschoben werden muss zurück. Dieser wird aber als **out**-Parameter angegeben (Methoden können in C# leider nur ein Objekt zurückgeben).

Der Algorithmus ist recht simpel, die Methode überprüft zuerst, welcher Teil des versperrenden Autos den Weg versperrt und speichert dies in **isX1**, einen Boolean, welcher wahr ist, wenn es das Heck ist (siehe Abb. 5, Zeile 10). Nun wird **isNullPos** mit 0 initialisiert. **isNullPos** gibt an, in welche Richtung es möglich ist, die Autos zu verschieben.

Nun wird in den beiden **for**-Schleifen nach dieser Position gesucht und in der zweiten überprüft, welche die beste ist und diese zurückgegeben.

In Abbildung 4 ist der Algorithmus zu sehen, welchen ich in dem Struktogramm oben versuchte zu visualisieren und zu erklären. Dieser muss nach dem Aufruf von **FindBestDirketion** nur noch die Autos einzeln verschieben. Und dass meine ich nicht nur sinnbildlich. Es ist nötig, da in **FindBestDirketion** nicht überprüft wird, um wie viele Schritte die Autos verschoben werden und so ist es möglich das ganze Schrittweise auszugeben. Ich entschied mich dafür, da es so leichter les- und verstehbar ist und es im dem Falle nicht auf Performanz ankommt.

4. Quellcode

```
private void start_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    try
    {
        string[] content = ReadFile();
        if (content.Length == 0) return;    // siehe ReadFile();

        Tuple<char, CarPart>[] parkingSpot = ConvertToTuple(ref content);

        string output = "";

        // rückwärts, damit's vorwärts ausgegeben wird
        for (int i = parkingSpot.Length - 1; i >= 0; i--)
        {
            output = ((char)(i + 65)) + ": ";

            // Wir müssen nur den Buchstaben des Autos ausgeben (davor steht ja keins)
            if (parkingSpot[i].Item1 == 0) Output(output, false);
            else
            {
                int break_pos;
                int add = FindBestDirektion(ref parkingSpot, out break_pos, ref i);

                string direktion_s = add == 1 ? "links" : "rechts";
                Tuple<char, CarPart>[] tempPS = new Tuple<char, CarPart>[parkingSpot.Length];
                parkingSpot.CopyTo(tempPS, 0);

                // von den angehaltenen Punkt auf-/abwärts
                for (int t = break_pos; t != i; t += add /* 1/-1 */)
                {
                    if (tempPS[t].Item1 == 0)  // wenn nicht leer, dann nächstes Auto nehmen
                    {
                        if (tempPS[t + add].Item1 == 0) // leer d.h. um zwei verschieben
                        {
                            tempPS[t] = tempPS[t + add * 2];             // fahre zwei Felder
                            tempPS[t + add] = tempPS[t + add * 3];

                            Clear(ref tempPS, t + add * 2);
                            Clear(ref tempPS, t + add * 3);

                            output += tempPS[t].Item1 + " zwei nach " + direktion_s + ", ";
                        }
                        else    // da steht was d.h. einmal verschieben
                        {
                            tempPS[t] = tempPS[t + add];    // fahre ein Feld
                            tempPS[t + add] = tempPS[t + add * 2];

                            Clear(ref tempPS, t + add * 2);

                            output += tempPS[t].Item1 + " eins nach " + direktion_s + ", ";
                        }

                    }
                }

                Output(output[..^2] /* entfernt das letzte ", " */, false);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Fehler!", MessageBoxButtons.OK, MessageBoxIcon.Question);
    }
```

Abb. 5: Code der „Hauptmethode“

```
private void Clear(ref Tuple<char, CarPart>[] cars, int index)
    { cars[index] = new Tuple<char, CarPart>((char)0, CarPart.Null); }

private Tuple<char, CarPart>[] ConvertToTuple(ref string[]  input)
    {. . .} /* konvertiert den gelesenen Inhalt in das 2er Tuple */

private int FindBestDirektion(
    ref Tuple<char, CarPart>[] parkingSpot, out int breakPos, ref int index
) {
    bool isX1 = parkingSpot[index].Item2 == CarPart.X1;
    int isNullPos = 0;
    breakPos = 0;

    for (int t = index; t < parkingSpot.Length; t++)
    {
        if (parkingSpot[t].Item1 == 0)
        {
            if (isX1 || isNullPos == 1) { breakPos = t; break; };
            isNullPos = 1;
        }
    }

    for (int t = index, stepCounter = index; t >= 0; t--, stepCounter++)
    {
        if (parkingSpot[t].Item1 == 0)
        {
            if (!isX1 || isNullPos < 0)
            {
                if (breakPos == 0 || breakPos > (isX1 ? stepCounter : stepCounter - 1))
                {
                    breakPos = t;
                    return 1;
                }
                return -1;
            }
            isNullPos -= 2;
        }
        else if (breakPos != 0 && stepCounter > breakPos + 1) return -1;
    }

    return -1;
}
```

Abb. 5: Code der restlichen relevanten Methoden

4. Beispiele

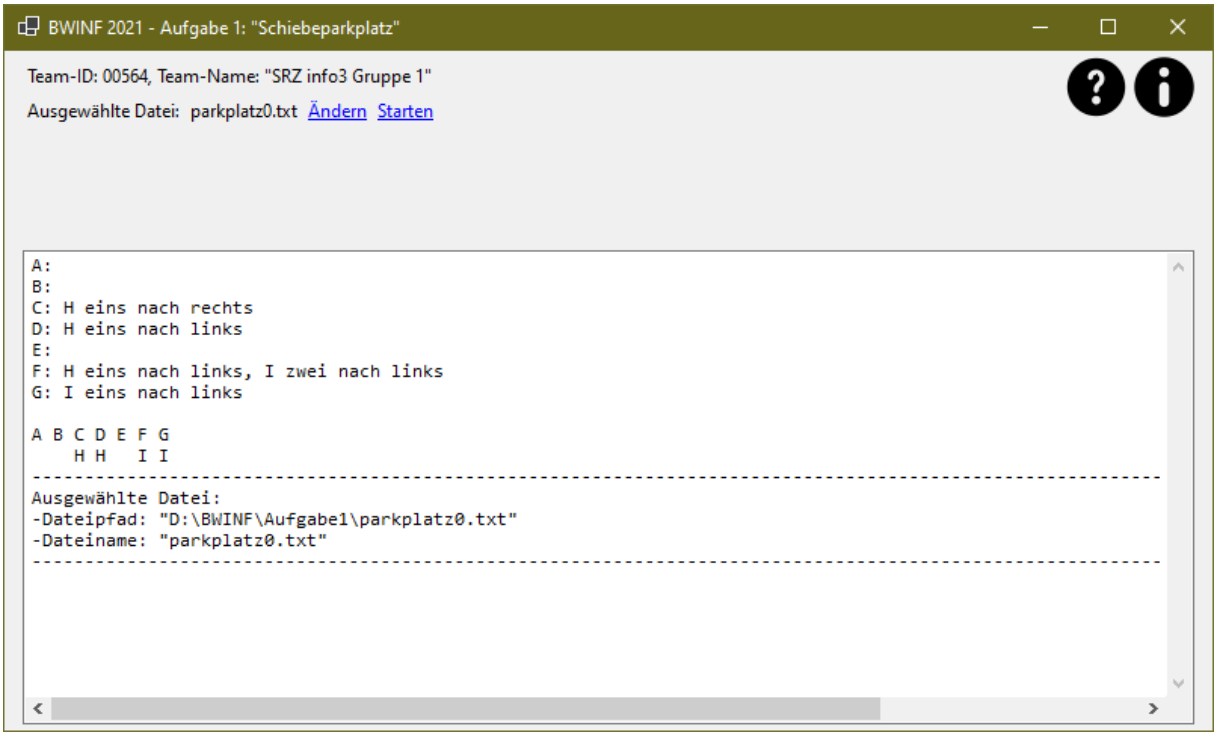

Abb. 6: Beispielparkplatz 0 mit UI

```
A:
B: P eins nach rechts, O eins nach rechts
C: O eins nach links
D: P eins nach rechts
E: O eins nach links, P eins nach links
F:
G: Q eins nach rechts
H: Q eins nach links
I:
J:
K: R eins nach rechts
L: R eins nach links
M:
N:

A B C D E F G H I J K L M N
  O O P P   Q Q     R R
-----------------------------------------------------------------------------------------------------------------------
Ausgewählte Datei:
-Dateipfad: "D:\BWINF\Aufgabe1\parkplatz1.txt"
-Dateiname: "parkplatz1.txt"
-----------------------------------------------------------------------------------------------------------------------
```

Abb. 7: Beispielparkplatz 1 im Ausgabefeld

```
A:
B:
C: O eins nach rechts
D: O eins nach links
E:
F: O eins nach links, P zwei nach links
G: P eins nach links
H: R eins nach rechts, Q eins nach rechts
I: P eins nach links, Q eins nach links
J: R eins nach rechts
K: P eins nach links, Q eins nach links, R eins nach links
L:
M: P eins nach links, Q eins nach links, R eins nach links, S zwei nach links
N: S eins nach links

A B C D E F G H I J K L M N
    O O   P P Q Q R R   S S
-----------------------------------------------------------------------------------------------------------------------
Ausgewählte Datei:
-Dateipfad: "D:\BWINF\Aufgabe1\parkplatz2.txt"
-Dateiname: "parkplatz2.txt"
-----------------------------------------------------------------------------------------------------------------------
```

Abb. 8: Beispielparkplatz 2 im Ausgabefeld

```
A:
B: O eins nach rechts
C: O eins nach links
D:
E: P eins nach rechts
F: P eins nach links
G:
H:
I: Q zwei nach links
J: Q eins nach links
K: Q zwei nach links, R zwei nach links
L: Q eins nach links, R eins nach links
M: Q zwei nach links, R zwei nach links, S zwei nach links
N: Q eins nach links, R eins nach links, S eins nach links

A B C D E F G H I J K L M N
  O O   P P     Q Q R R S S
-------------------------------------------------------------------------------------------------------------------------------------------------------
Ausgewählte Datei:
-Dateipfad: "D:\BWINF\Aufgabe1\parkplatz3.txt"
-Dateiname: "parkplatz3.txt"
-------------------------------------------------------------------------------------------------------------------------------------------------------
```

Abb. 9: Beispielparkplatz 3 im Ausgabefeld

```
A: R eins nach rechts, Q eins nach rechts
B: R zwei nach rechts, Q zwei nach rechts
C: R eins nach rechts
D: R zwei nach rechts
E:
F:
G: S eins nach rechts
H: S eins nach links
I:
J:
K: T eins nach rechts
L: T eins nach links
M:
N: U eins nach rechts
O: U eins nach links
P:

A B C D E F G H I J K L M N O P
Q Q R R     S S     T T   U U
-------------------------------------------------------------------------------------------------------------------------------------------------------
Ausgewählte Datei:
-Dateipfad: "D:\BWINF\Aufgabe1\parkplatz4.txt"
-Dateiname: "parkplatz4.txt"
-------------------------------------------------------------------------------------------------------------------------------------------------------
```

Abb. 10: Beispielparkplatz 4 im Ausgabefeld

```
A:
B:
C: P zwei nach links
D: P eins nach links
E: Q eins nach rechts
F: Q zwei nach rechts
G:
H:
I: R eins nach rechts
J: R eins nach links
K:
L:
M: S eins nach rechts
N: S eins nach links
O:

A B C D E F G H I J K L M N O
    P P Q Q     R R     S S
-------------------------------------------------------------------------------------------------------------------------------------------------------
Ausgewählte Datei:
-Dateipfad: "D:\BWINF\Aufgabe1\parkplatz5.txt"
-Dateiname: "parkplatz5.txt"
-------------------------------------------------------------------------------------------------------------------------------------------------------
```

Abb. 11: Beispielparkplatz 5 im Ausgabefeld

5. Quellen

Das Hilfe-Icon habe ich von hier:
https://icon-icons.com/de/symbol/helfen-badged-gef%C3%BCllt/142888

Und das Info-Icon von hier:
https://icon-icons.com/de/symbol/info-Informationen/142931

Wie in Junioraufgabe 1 half mir das bei der Erstellung des Struktogramms (Abb. TODO) folgender Link: https://dditools.inf.tu-dresden.de/ovk/Informatik/Programmierung/Grundlagen/Struktogramme.html

Ferner ist der Code und der Rest unserer BwInf-Lösungen auf GitHub:
https://github.com/Metis-Git/BwInf-2021/