Aufgabe 3: Rotation

Lösungsidee:

Das Puzzle besteht aus gleich vielen Reihen und Spalten und ähnelt daher vom Aufbau sehr einer Matrix. Diese Matrix wird dementsprechend beim Drehen nach links um 90° nach links und beim Drehen nach rechts um 90° nach rechts gedreht. Nach dem Drehen müssen nun noch alle Blöcke in dem Puzzle der Gravitation entsprechend nach unten fallen.
Da beim Verdrehen des Ausgangszustandes immer genau zwei neue Zustände entstehen können(links-/Rechtsdrehen), können die Verdrehungen als Binärbaum abgebildet werden.
Um den Lösungsraum einzugrenzen, müssen alle bisherigen Verdrehungen abgespeichert und mit allen neuen Verdrehungen verglichen werden. Wenn eine Verdrehung mit einer schon einmal aufgetretenen identisch ist, kann die Suche an diesem Blatt des Baumes abgebrochen werden.
Alle noch zu überprüfenden Verdrehungen müssen sich auch gemerkt werden. Nach jedem Verdrehen bei dem eine Konfiguration des Puzzles entsteht, die noch nie aufgetreten ist, muss diese gespeichert werden.
Da bei dem Erstellen des Baumes jedes Level komplett aufgebaut wird, bevor ein neues Level erstellt wird, ist die erste Lösung, die gefunden wird, auch eine optimale Lösung.

Umsetzung:

Die Lösungsidee wird in C# implementiert.
Für die Matrix wird eine eigene Klasse erstellt. Diese Klasse hat folgende Attribute: ein 2D-Array aus Char für den Inhalt, einen Integer für die Größe und Methoden für das Ausgeben, für das Drehen, für das Simulieren der Schwerkraft und einen Comparer zum Vergleichen.
Der Binärbaum wird auch als eine Klasse dargestellt.
Diese Klasse hat vier Listen jede dieser Listen speichert die Verdrehungen in denen die Öffnung in eine bestimmte Richtung(oben, rechts, unten und links) zeigt und auch die Startverdrehung der Matrix.
Jeder Knoten des Baumes wird als eine Instanz der Klasse Node dargestellt.
Diese Klasse Node hat fünf Eigenschaften: den vorherigen Knoten, die jetzige Verdrehung des Puzzles als Matrix, die Tiefe des jetzigen Knoten und noch die Richtung, in die als letztes gedreht wurde.
Beim Start des Programms wird eine Datei eingelesen und der Startknoten für den Baum daraus erstellt.
Daraufhin wird immer die jetzige Ausrichtung der Öffnung des Puzzles überprüft und mit allen bisher bekannten Puzzeln dieser Verdrehung verglichen. Da eine lineare Suche bei dem sehr komplexen dritten Beispiel zu langsam ist, muss eine binäre Suche hierfür verwendet werden.
Wenn kein identisches Puzzle hierbei gefunden wird, wird diese Verdrehung mit in der Liste abgespeichert.
Wenn ein identisches Puzzle gefunden wird, ist diese Verdrehung hinfällig und wird gelöscht und die nächste Verdrehung wird untersucht.
Dieser Vorgang wird solange wiederholt, bis entweder ein Block aus dem Puzzle fällt oder es keine Verdrehung mehr gibt, die noch nie aufgetreten ist.
Wenn eine Lösung gefunden wurde, wird von der Startverdrehung (S →) aus alle Anweisungen ausgegeben(links Drehen - l und rechts Drehen – r), um das Puzzle zu lösen.

Quelltext:

Die Funktion um die Gravitation auf die Matrix zu wirken:

```
public bool ApplyGravity(char[,] _content)
        {
            //ob eine Loesung gefunden wurde
            bool foundSolution = false;

            last = size - 1;
            //Console.WriteLine("");
            //Anfang bei der untersten Reihe
            for (int y = this.last - 1; y > 0; y--)
            {
                //ganz links
                for (int x = 1; x < this.last; x++)
                {
                    //wenn dieses Feld eine Zahl ist
                    if (char.IsNumber(_content[y, x]))
                    {
                        //wenn der Block horizontal ausgerichtet ist
                        //ob an keiner Stelle ein BLock darunter ist
                        bool isMoveable = false;
                        //wenn der Block unter diesem Feld frei ist
                        //Und es nicht ein Teil aus dem Rahmen ist
                        //ODER
                        if (_content[y + 1, x] == ' ')// && y + 1 != last )
                            isMoveable = true;

                        //die Ziffer fuer den momentanigenBlock
                        char type = _content[y, x];


                        //die Laenge des Blockes
                        int length = 1;
                        for (int right = x + 1; right < last; right++)
                        {
                            //wenn rechts daneben ein Block  ist
                            if (_content[y, right] == type)
                                length++;
                            else
                                break;
                            //wenn das Feld darunter nicht frei ist
                            //UND das Feld rechts daneben vom selben typ ist
                            if (_content[y + 1, right] != ' ' && _content[y, right + 1] == type)
                                isMoveable = false;

                            if (!isMoveable && _content[y, right] == type)
                                x = right;
                        }
                        if (isMoveable)
                        {
                            for (int down = y; down < last; down++)
                            {
                                //ueber die Laenge des jetztigen Blockes
                                //jeden einzelnen Stein des BLockes ueberpruefen ob er nach unten bewegbar
                                ist
                                for (int m = 0; m < length; m++)
                                {
                                //wenn der Block darunter keine # ist UND (vom gleichen Typ ist ODER ein
                                freies Feld ist)
                                    if (_content[down + 1, x + m] != '#' && (_content[down + 1, x + m] ==
                                        type || _content[down + 1, x + m] == ' '))
                                        isMoveable = true;
                                    else
                                    {
                                        isMoveable = false;
                                        break;
                                    }
                                }
                                //wenn der gesamte Block noch bewegbar ist
                                if (isMoveable)
                                {
                                    //ihn ueber die gesamte laenge verschieben
                                    for (int i = 0; i < length; i++)
                                    {
```

```
                                //tauscht den Block mit dem darunter
                                Swap(ref _content[down, x + i], ref _content[down + 1, x + i]);
                                //wenn der letzte Block(der im Rhamen) eine Zahl ist
                                //-->  es ist eine Loesung gefunden
                                if (down + 1 == last && _content[down + 1, x] == type)
                                {
                                    foundSolution = true;
                                }
                            }
                        }
                        else
                            break;
                    }
                }
            }
        }
    }
    return foundSolution;
}
```

Beispiele:

Alle Beispieleingaben sind lösbar und werden innerhalb von wenigen Sekunden berechnet.

Rotation1: S → llllrl

Rotation2: S → rrrrlrrlllllllrrrrrrlr

Rotation3: S → lrrrrlllrrrllllllllrrllrrrrrrrlrrrrrrrrrrrrlllrrrrrllllrrrrllrrrrrrrrlrllllllllllllrllrrrrrlr