

Aufgabe 1: Nandu

Team-ID: 00206

Team-Name: Julian

Bearbeiter/-innen dieser Aufgabe:
Julian Bents

19. November 2023

Inhaltsverzeichnis

1	Lösungsidee	1
2	Umsetzung	1
3	Beispiele	2
4	Quellcode	6

1 Lösungsidee

Mein Ansatz war, die Aufgabe als Simulation zu modellieren. Dabei werden verschiedene Bausteine verwendet, die alle identisch sind, aber unterschiedliche Verhaltensweisen aufweisen. Die Simulation wird für alle möglichen Kombinationen durchgeführt, und die Ergebnisse können anschließend gespeichert werden.

2 Umsetzung

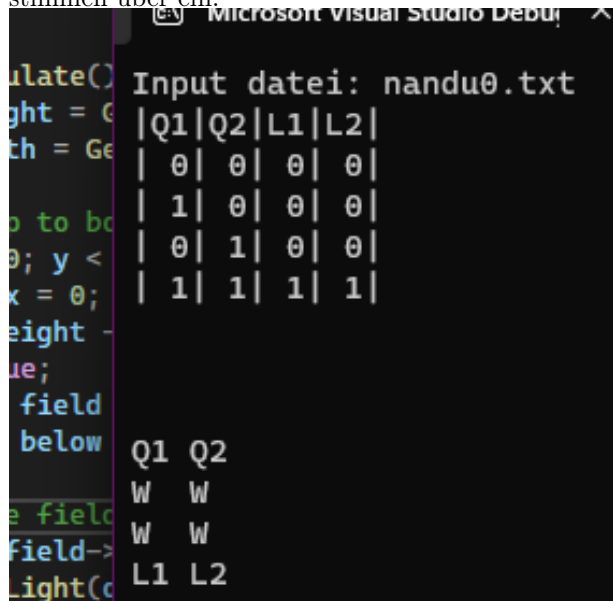
Für jeden einzelnen Baustein habe ich eine Klasse erstellt, die von der abstrakten Klasse `Feld` abgeleitet ist. Dadurch war es möglich, ein 2D-Array von Feldern zu erstellen, das die gesamte Konstruktion repräsentiert. Jede Klasse speichert Informationen darüber, ob sie bestrahlt wird, und kann berechnen, ob sie das darunterliegende Feld beleuchten soll. Auf diese Weise kann die gesamte Konstruktion simuliert werden, indem das Array von links nach rechts und von oben nach unten durchlaufen wird. Dabei wird ermittelt, ob das Licht von dem aktuellen Feld aktiviert ist, und falls ja, wird dem darunterliegenden Feld mitgeteilt, dass es bestrahlt wird.

Allerdings stellte sich ein Problem heraus, da die weißen, roten und blauen Bausteine sich über zwei Felder erstrecken. Dies wurde jedoch durch Anpassungen in den Verhaltensmethoden gelöst. Die Bausteine wurden so modifiziert, dass sie wissen, zu welchem Baustein sie gehören. Dabei konnte ich effektiv shared pointers nutzen.

Das Programm arbeitet dann folgendermaßen: Zunächst werden alle möglichen Kombinationen von Lampen generiert. Anschließend wird für jede dieser Möglichkeiten die Simulation durchgeführt. Am Ende werden alle Ergebnisse auf der Konsole ausgegeben.

3 Beispiele

In folgenden ist das Programm für das linke Beispiel der Aufgabe ausgeführt worden, die Ergebnisse stimmen überein:



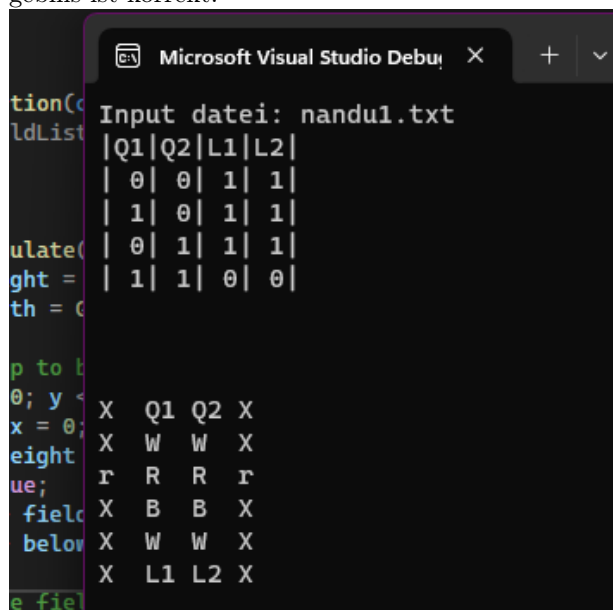
```

Input datei: nandu0.txt
|Q1|Q2|L1|L2|
| 0| 0| 0| 0|
| 1| 0| 0| 0|
| 0| 1| 0| 0|
| 1| 1| 1| 1|

Q1 Q2
W W
W W
L1 L2

```

In folgenden ist das Programm für das rechte/erste Beispiel der Aufgabe ausgeführt worden, das Ergebnis ist korrekt:



```

Input datei: nandu1.txt
|Q1|Q2|L1|L2|
| 0| 0| 1| 1|
| 1| 0| 1| 1|
| 0| 1| 1| 1|
| 1| 1| 0| 0|

X Q1 Q2 X
X W W X
r R R r
X B B X
X W W X
X L1 L2 X

```

In folgenden ist das Programm für das zweite Beispiel der Aufgabe ausgeführt worden:

```

Microsoft Visual Studio Debug Console
Input datei: nandu2.txt
|Q1|Q2|L1|L2|
| 0| 0| 0| 1|
| 1| 0| 0| 1|
| 0| 1| 0| 1|
| 1| 1| 1| 0|

X  X  Q1 Q2 X  X
X  X  B  B  X  X
X  X  W  W  X  X
X  r  R  R  r  X
r  R  W  W  R  r
X  W  W  B  B  X
X  X  B  B  X  X
X  X  L1 L2 X  X

```

In folgenden ist das Programm für das dritte Beispiel der Aufgabe ausgeführt worden:

```

Microsoft Visual Studio Debug Console
Input datei: nandu3.txt
|Q1|Q2|Q3|L1|L2|L3|L4|
| 0| 0| 0| 1| 0| 0| 1|
| 1| 0| 0| 0| 1| 0| 1|
| 0| 1| 0| 1| 0| 1| 1|
| 1| 1| 0| 0| 1| 1| 1|
| 0| 0| 1| 1| 0| 0| 0|
| 1| 0| 1| 0| 1| 0| 0|
| 0| 1| 1| 1| 0| 1| 0|
| 1| 1| 1| 0| 1| 1| 0|

X  X  Q1 Q2 X  X  Q3 X
X  X  B  B  X  r  R  X
X  r  R  R  r  W  W  X
X  W  W  B  B  W  W  X
r  R  B  B  B  B  R  r
X  B  B  W  W  B  B  X
X  L1 L2 X  L3 L4 X  X

```

In folgenden ist das Programm für das vierte Beispiel der Aufgabe ausgeführt worden:

```

Microsoft Visual Studio Debug Console
Input datei: nandu4.txt
|Q1|Q2|Q3|Q4|L1|L2|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

X X Q1 Q2 Q3 Q4 X X
X r R B B R r X
X X W W W W X X
X r R B B R r X
r R W W B B R r
X W W W W W W X
X X B B X X X X
X X L1 L2 X X X X
  
```

In folgenden ist das Programm für das fünfte Beispiel der Aufgabe ausgeführt worden, ich habe dies ausnahmsweise nicht auf Richtigkeit geprüft, die Muster in den Ergebnissen geben mir allerdings Vertrauen ;)

X	X	X	X	X	Q1	Q2	X	X	X	Q3	Q4	X
X	X	X	X	X	R	R	R	X	X	r	R	X
X	X	X	X	X	r	R	r	r	R	R	r	
X	X	X	X	W	W	B	B	W	W	B	B	
X	X	X	r	R	B	B	B	B	B	B	B	
X	X	r	R	B	B	X	B	B	B	B	B	
X	r	R	B	B	R	r	B	B	B	B	B	
X	X	B	B	W	W	B	B	X	B	B	B	
X	r	R	W	W	W	W	R	r	W	W	W	
X	W	W	B	B	W	W	X	B	B	W	W	
r	R	B	B	B	B	W	r	X	B	B	B	
X	B	B	B	B	B	B	R	r	B	B	B	
r	R	B	B	B	B	B	B	B	B	B	B	
L1	X	X	X	X	L2	X	X	X	X	L3	X	

4 Quellcode

Die Main Funktion des Programmes, hier wird der Input genommen, dann wird die Simulation für jede mögliche Kombination ausgeführt und am Ende werden alle Ergebnisse auf der Konsole ausgegeben:

```

1 // structure to store simulation result
2 struct SimulationData {
3     std::vector<std::pair<uint32_t, bool>> flashlights;
4     std::vector<std::pair<uint32_t, bool>> leds;
5 };
6 [...]
7 int main() {
8     // write user input into 2D array
9     std::vector<std::vector<Ref<Field>>> fields;
10    TakeUserInput(fields);
11
12    Construction constr(fields);
13
14    // get a list to all flashlights and leds
15    auto flashLights = constr.GetAllOfType(FieldType::Flashlight);
16    auto leds = constr.GetAllOfType(FieldType::LED);
17
18    std::vector<SimulationData> combinations;
19
20    // calculate the total number possible of combinations (2^n)
21    uint32_t totalCombinations = 1 << (uint32_t)flashLights.size();
22
23    for (uint32_t i = 0; i < totalCombinations; i++) {
24        SimulationData data;
25        // reset the simulation
26        constr.Reset();
27
28        // using a binary representation we can generate the current combinations
29        for (uint32_t j = 0; j < flashLights.size(); j++) {
30            // extract the j'th bit
31            bool value = (i >> j) & 1;
32            // save the value to be printed later and set it in the construction
33            data.flashlights.push_back({ j, value });
34            flashLights[j]->SetLight(value);
35        }
36        constr.Simulate();
37
38        // save the reusults
39        for (uint32_t j = 0; j < leds.size(); ++j) {
40            data.leds.push_back({ j, leds[j]->IsLightOn() });
41        }
42        combinations.push_back(data);
43    }
44
45    PrintResults(combinations, constr);
46
47
48    return 0;
49 }

```

Teile der Field.h Datei, hier werden alle Baustein und deren Regeln definiert:

```

1 class Field {
2 public:
3     virtual ~Field() = default;
4     virtual void SetLight(bool value) = 0;
5     virtual bool IsLightOn() const = 0;
6     virtual FieldType GetType() const = 0;
7 };
8
9 class EmptyField : public Field {
10 [...]

```

```

11 };
12 class FlashlightField : public Field {
13 [...]
14 };
15 class LEDField : public Field {
16 [...]
17 };
18 // for fields that are in pairs
19 class BlockField : public Field {
20 [...]
21 };

22
23 class WhiteField : public BlockField {
24 public:
25     WhiteField() {}
26     virtual ~WhiteField() = default;
27
28     // Die LEDs leuchten immer beide, ausser wenn beide Sensoren bestrahlt werden
29     virtual bool IsLightOn() const override {
30         return !(this->IsReceiving() && GetOtherPair()->IsReceiving());
31     }
32
33     virtual FieldType GetType() const override {
34         return FieldType::White;
35     }
36 private:
37 };

38
39 class RedField : public BlockField {
40 public:
41     RedField(bool isSensor)
42         : m_isSensor(isSensor) {}
43
44     virtual ~RedField() = default;
45
46     // Beide LEDs strahlen dann, wenn kein Licht auf den Sensor faellt
47     virtual bool IsLightOn() const override {
48         if (m_isSensor) {
49             return !m_receiving;
50         }
51         Ref<RedField> other = std::dynamic_pointer_cast<RedField>(GetOtherPair());
52         return other->IsLightOn();
53     }
54
55     virtual FieldType GetType() const override {
56         return FieldType::Red;
57     }
58     inline bool IsSensor() { return m_isSensor; };
59 private:
60     bool m_isSensor = false;
61 };

62
63 class BlueField : public BlockField {
64 public:
65     BlueField() {}
66     virtual ~BlueField() = default;
67
68     // Bei den blauen Bausteinen wird das Licht einfach weitergegeben
69     virtual bool IsLightOn() const override { return m_receiving; }
70
71     virtual FieldType GetType() const override {
72         return FieldType::Blue;
73     }
74 };

```

Hier ist der Code der eigentlichen Simulation zu finden:

```

void Construction::Simulate() {
2     const uint32_t height = GetHeight();
3     const uint32_t width = GetWidth();
4
5     // iterate from top to bottom, left to right

```

```
6  for (uint32_t y = 0; y < height; y++) {
    for (uint32_t x = 0; x < width; x++) {
8      if (y >= height - 1)
          continue;
10     Ref<Field> field = At(x, y);
        Ref<Field> below = At(x, y + 1);
12
        // tell the field below me if its lit up
14     bool on = field->IsLightOn();
        below->SetLight(on);
16     }
18 }
```