|  |
| --- |
| Dienststelle Informatik  Ruopigenplatz 1 6015 Luzern Telefon 041 228 56 15 Telefax 041 228 59 56 informatik@lu.ch www.informatik.lu.ch |
|



Ken Häsler

17.01.2021

Analyse NIM-Spiel

HFIE 2021 – Programmierauftrag

Änderungsverzeichnis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Datum** | **Status** | **Autor** | **Kommentar** |
| 0.1 | 17.01.2021 | Entwurf | Ken Häsler | Entwurf |

Tabelle 1: Änderungsverzeichnis

Verteiler

* Rolf Laich (Lehrperson)

Zweck des Dokuments

Dieses Dokument schafft eine Übersicht über die vollzogene praktische Arbeit. Ausgangslage, Informations- sowie Wissensbeschaffung, Entscheide, Umsetzung, Kontrollen und Auswertungen werden hier dokumentiert.

Inhalt

[Teil 1](#_Toc61801708)

[1 Aufgabenstellung 5](#_Toc61801709)

[1.1 Ausgangslage 5](#_Toc61801710)

[1.2 Detaillierte Aufgabenstellung 5](#_Toc61801711)

[1.2.1 Anforderungen 5](#_Toc61801712)

[1.3 Vorgaben 5](#_Toc61801713)

[1.4 User Input 5](#_Toc61801714)

[1.5 Vorgehen 6](#_Toc61801715)

[1.5.1 Analyse 6](#_Toc61801716)

[1.5.2 Implementierung 6](#_Toc61801717)

[1.5.3 Source Code Verwaltung und Hilfestellungen 6](#_Toc61801718)

[1.6 Bewertung 6](#_Toc61801719)

[Teil 2](#_Toc61801720)

[1 Analyse 7](#_Toc61801721)

[1.1 Grundüberlegungen 7](#_Toc61801722)

[2 UML-Diagramm 8](#_Toc61801723)

[3 Flussdiagramm 9](#_Toc61801724)

[4 Funktionen 10](#_Toc61801725)

[4.1 CreateRandomNimNum 10](#_Toc61801726)

[4.2 ShowNimGame 10](#_Toc61801727)

[4.3 NewMove 11](#_Toc61801728)

[4.4 AskPlayer 11](#_Toc61801729)

[4.5 CheckGameStatus 11](#_Toc61801730)

[5 Testing 12](#_Toc61801731)

[5.1 Vorarbeiten 12](#_Toc61801732)

[5.2 ShowNimGame 12](#_Toc61801733)

[5.2.1 Testmotiv 12](#_Toc61801734)

[5.2.2 Mittel und Methoden 12](#_Toc61801735)

[5.2.3 Erwartungen 12](#_Toc61801736)

[5.2.4 Resultat 12](#_Toc61801737)

[5.3 NewMove 13](#_Toc61801738)

[5.3.1 Testmotiv 13](#_Toc61801739)

[5.3.2 Mittel und Methoden 13](#_Toc61801740)

[5.3.3 Erwartungen 13](#_Toc61801741)

[5.3.4 Resultat 13](#_Toc61801742)

[5.4 AskUser 13](#_Toc61801743)

[5.4.1 Testmotiv 13](#_Toc61801744)

[5.4.2 Mittel und Methoden 14](#_Toc61801745)

[5.4.3 Erwartungen 14](#_Toc61801746)

[5.5 Schlussfolgerung 15](#_Toc61801747)

[6 Schlusswort 16](#_Toc61801748)

[7 Abkürzungsverzeichnis 17](#_Toc61801749)

[8 Glossar 17](#_Toc61801750)

[9 Abbildungsverzeichnis 18](#_Toc61801751)

[10 Tabellenverzeichnis 19](#_Toc61801752)

[11 Codeverzeichnis 20](#_Toc61801753)

[12 Quellenverzeichnis 21](#_Toc61801754)

[13 Anhang 22](#_Toc61801755)

Teil 1

# Aufgabenstellung

## Ausgangslage

NIM ist ein Strategiespiel, das von zwei Spielern gespielt wird. Es werden Streichhölzer in mehreren Reihen angeordnet. Abwechslungsweise nehmen jetzt die Spieler Hölzchen weg. Ein Spieler muss ein oder mehrere Hölzchen nehmen. Es dürfen aber nur Hölzchen aus einer einzelnen Reihe genommen werden.

Derjenige Spieler, welcher das oder die letzten Hölzchen nehmen kann, hat gewonnen.

## Detaillierte Aufgabenstellung

Ein NIM Spiel soll als Console-Applikation entwickelt werden. Dabei sollen diese Anforderungen berücksichtigt werden:

### Anforderungen

* **Initialisierung des Spiels**: Die Anzahl Reihen und Anzahl Hölzchen können entweder vom Benutzer eingegeben werden oder vom Programm zufällig erzeugt werden. Beide Varianten sind zulässig.
* **Anzeige**: Der aktuelle Stand (Anzahl Hölzchen pro Reihe) muss in verständlicher Form auf der Konsole angezeigt werden. Wenn das Spiel zu Ende ist, soll der Gewinner angezeigt werden.
* **Spielverlauf:** Der Spieler wechselt sich ab mit dem Computer. Wenn der Spieler an der Reihe ist, wird er über eine Meldung aufgefordert, seinen Zug zu machen.
* **Benutzer-Spielzug:** Der Benutzer kann eingeben, in welcher Zeile, wie viele Hölzchen entfernt werden sollen. Fehlerhafte Eingaben des Benutzers sollen das Programm nicht zum Abstürzen bringen.
* **Computer-Spielzug:** Der Computer kann einen gültigen Spielzug durchführen. Es kann (muss aber nicht, gibt aber Zusatzpunkte) eine Gewinn-Strategie implementiert werden. Vergl. <https://www.hep-verlag.ch/pub/media/import/public/6486/mathematikaufgaben.pdf>. Auch verschieden «Levels» sind denkbar.
* **Korrektheit:** Die Spielzüge müssen korrekt angewendet werden.
* **Usability:** Mit Hilfe der angezeigten Meldungen soll dem Benutzer klar sein, wie das Spiel zu benutzen ist.

## Vorgaben

Für die Implementierung des Spiels können diese Header verwendet werden:



Code 1: Header-Bibliotheken

## User Input

Für das Lesen von Benutzereingaben kann die Funktion scanf\_s verwendet werden:



Code 2: Funktion Scanf

## Vorgehen

Bei der Umsetzung der Arbeit soll wie folgt vorgegangen werden:

### Analyse

* Zeichnen Sie ein UML Aktivitäts-Diagramm, welches den Ablauf und die einzelnen Schritte des Spiels zeigt. Überlegen Sie sich, welche Aktionen sie in weitere Aktionen aufteilen möchten.
* Überlegen Sie sich, welche Daten Sie brauchen, um den Zustand des Spiels festzuhalten.
* Definieren Sie eine Reihenfolge, in welcher Reihenfolge Sie die Funktionen implementieren und testen.
* Definieren Sie, wie/ob Sie die einzelnen Funktionen testen (z.B. Bei der Anzeigefunktion sehen sie sofort, wenn etwas nicht stimmt. Das könne Sie so festhalten und sagen, dass sie das nicht ausdrücklich testen)

### Implementierung

* Definieren Sie für die alle Aktionen c-Funktionen
* Definieren Sie geeignete Datenstrukturen
* Definieren Sie die entsprechenden Variablen, um den Spielstatus zu speichern.
* Implementieren und testen Sie die Funktionen gemäss Ihrem Plan.

### Source Code Verwaltung und Hilfestellungen

* Speichern Sie das Projekt mit git (github oder gitlab) und geben Sie mir Zugriff auf dieses Repository. Ich werde Kommentare, Anmerkungen u.U. Implementierungsvorschläge in einem Branch zu diesem Projekt machen.
* Teilen Sie mir mit, wenn Sie die Analyse abgeschlossen habe. Ich kann ihnen dann eine Rückmeldung dazu geben. Es ist aber normal, dass man während der Implementierung feststellt, dass eine Überlegung aus der Analyse so nicht funktioniert. Dennoch macht es Sinn, mit einer Analyse zu beginnen.
* Ich werde Sie bei der Lösung dieser Aufgabe unterstützen. Es sollen aber von ihnen klare eindeutige Fragen kommen.

## Bewertung

Bewertet wird die Erfüllung der Anforderungen 70% sowie die Analyse 30%. Die gesamte Arbeit ergibt eine Note mit Gewicht 40%.

Teil 2

# Analyse

## Grundüberlegungen

Um mir einen groben Überblick über meinen Auftrag zu verschaffen, habe ich mir zu Beginn der Arbeit überlegt, was für Funktionen mein Programm benötigen könnte:

|  |  |
| --- | --- |
| **Funktion** | **Zweck** |
| Main | Hauptfunktion |
| ShowGame | Anzeigen des aktuellen Spiels |
| GenerateRandom | Zufälliges Spiel generieren |
| AskUser | User-Input |
| NewMove | Neuer Spielzug |
| CheckGameStatus | Status des Spiels abfragen |

Tabelle 2: NIM-Funktionen

Im Anschluss dazu habe ich mich gefragt, wie es möglich wäre den aktuellen Spielstand nicht nur einfach anzeigen zu lassen, sondern auch Spielzüge einfach miteinfliessen lassen zu können.

Aufgrund des Aufbaus des NIM-Spiels habe ich mich schlussendlich für ein eindimensionales Integer-Array entschieden, bei welchem die Streichhölzchen lediglich visuell implementiert werden.

Mithilfe von zwei For-Schlaufen, wird dann für die Anzahl jeder Stelle im Array ein «I» ausgegeben.

Folgend ein Beispiel zu diesem Array.

|  |  |
| --- | --- |
| **Array-Index** | **Wert** |
| 0 | 2 |
| 1 | 8 |
| 2 | 5 |
| 3 | 1 |

Tabelle 3: NIM-Spiel Array

Folgend ein Beispiel zur Konsolen-Ansicht:

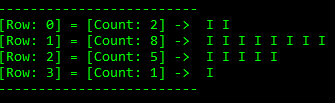


Abbildung 1: NIM-Spielansicht

# UML-Diagramm

Folgendes UML-Aktivitätsdiagramm wurde für die Konsolen-Applikation definiert:

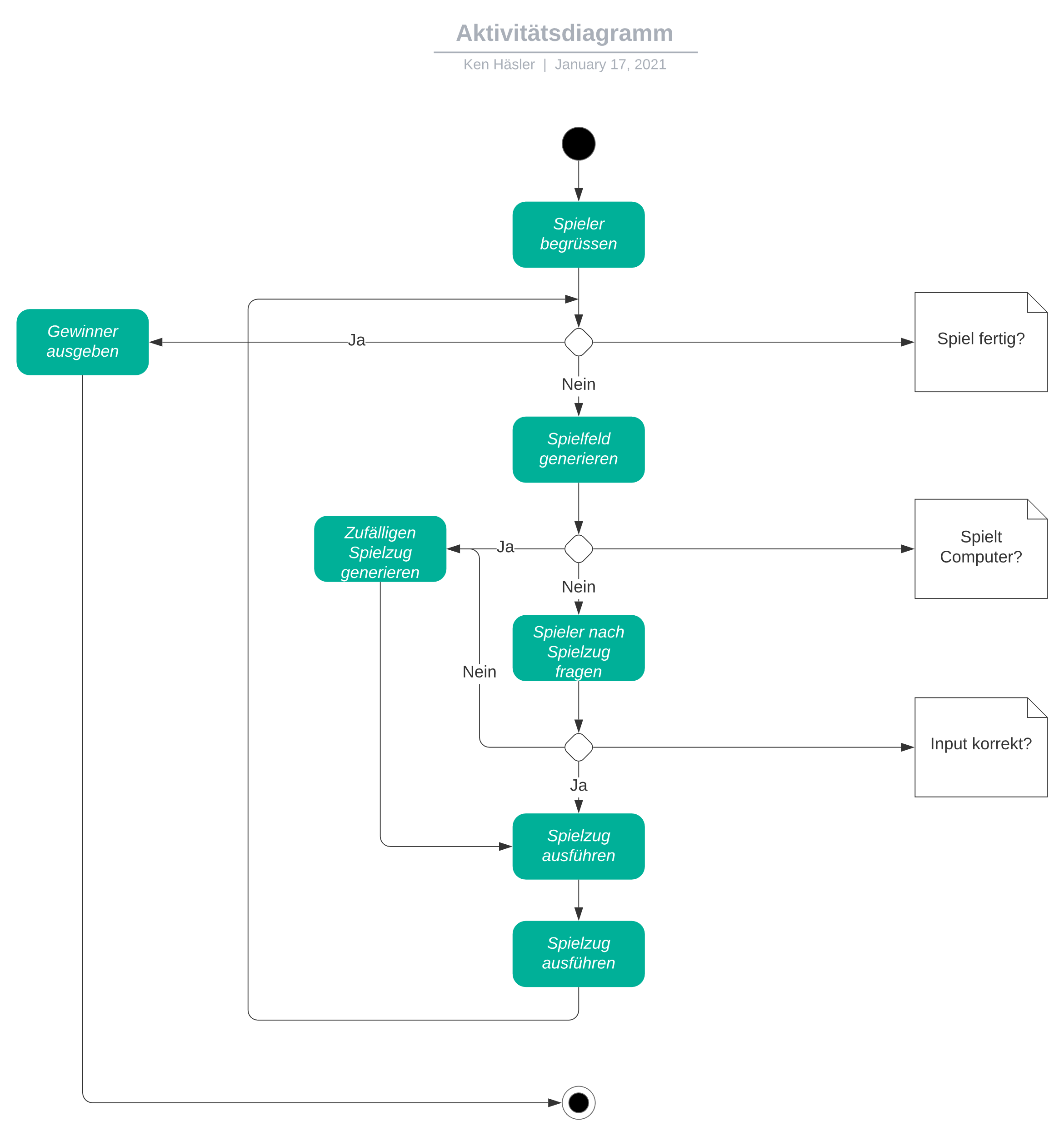


Abbildung 2: UML-Aktivitätsdiagramm

Zu Beginn wird der Spieler von dem Programm begrüsst. Im Anschluss dazu wird das Spielfeld generiert und der Spieler nach seinem Zug gefragt. Sollte der gewählte Spielzug nicht möglich sein, setzt der Spieler einen Zug aus. Wenn der Input korrekt eingegeben wurde, wird der Spielzug ausgeführt und der Computer ist mit seinem Zug an der Reihe. Nach jedem Zug wird überprüft, ob das Spiel bereits gewonnen oder verloren wurde. Der jeweilige Gewinner wird zum Schluss des Spiels ausgegeben.

# Flussdiagramm

Folgendes Flussdiagramm wurde für die Konsolen-Applikation definiert:

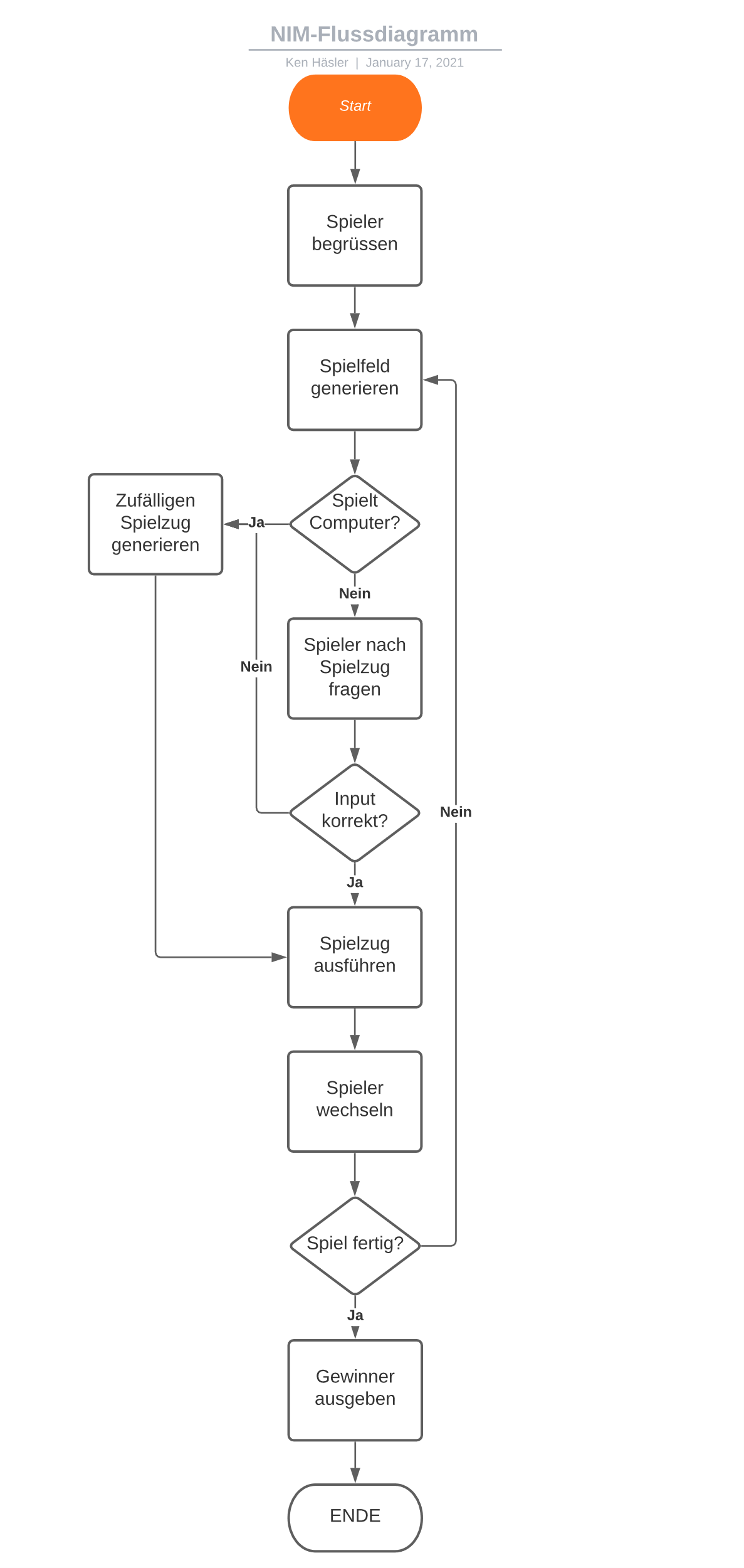


Abbildung 3: Flussdiagramm

# Funktionen

Folgende Funktionen wurden in diesem Ablauf für das C-Programm definiert:

* void CreateRandomNimNum(int nimdata[]);
* void ShowNimGame(const int nimdata[]);
* void NewMove(int\* row, int\* num, const int player, int nimdata[]);
* void AskPlayer(int\* row, int\* num);
* int CheckGameStatus(const int nimdata[]);

## CreateRandomNimNum



Code 3: Funktion CreateRandomNimNum

## ShowNimGame



Code 4: Funktion ShowNimGame

## NewMove



Code 5: Funktion NewMove

## AskPlayer



Code 6: Funktion AskPlayer

## CheckGameStatus



Code 7: Funktion CheckGameStatus

# Testing

## Vorarbeiten

Um die verschiedenen Funktionen meines C-Programms schnell und einfach testen zu können, habe ich im Programm einen Test-Modus hinterlegt, welcher durch das Abändern eines «Define-Statements» aktiviert werden kann.

Effektiv getestet werden die einzelnen Funktionen.

## ShowNimGame

### Testmotiv

* Wird das Spiel korrekt angezeigt

### Mittel und Methoden

Getestet wird mithilfe der Funktion, sowie mittels Ausgabe durch «printf».

### Erwartungen

Das Spiel wird mit der Übergabe des Spiel-Arrays korrekt angezeigt.

### Resultat

Das Spiel wird korrekt angezeigt und gibt dem Benutzer eine Hilfestellung zur Eingabe.

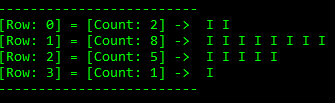


Abbildung 4: Test Spielansicht

## NewMove

### Testmotiv

* Kann der Spielzug für den Benutzer ausgeführt werden
* Kann der Spielzug für den Computer ausgeführt werden

### Mittel und Methoden

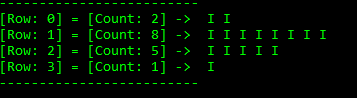
Getestet wird durch das Ausführen des kompletten Programms und der Überprüfung der Spielzüge durch die ShowNimGame Funktion.

### Erwartungen

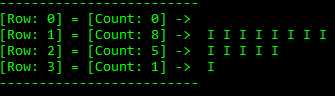
Spielzüge werden korrekt ausgeführt und angezeigt.

### Resultat

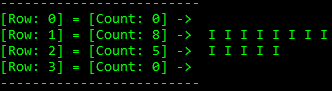
Vor den Spielzügen:



Nach Spielzug des Spielers:



Nach Spielzug des Computers:



## AskUser

### Testmotiv

Um herausfinden zu können, ob der User-Input richtig verarbeitet wird, müssen zwei Sachen getestet werden.

* Werden die Variablen richtig übergeben
* Werden Fehl-Eingaben des Benutzers abgefangen

### Mittel und Methoden

Um die oben genannten Fälle testen zu können, wurden zwei Test-Abläufe temporär im Code implementiert.

**Erster Test:**

**Zweiter Test:**

### Erwartungen

Ich erwarte, dass der Benutzer keine Fehleingaben machen kann, sowie die Variablen konform an das Hauptprogramm (Main) übergeben werden.

## Schlussfolgerung

Die getesteten Bereiche schnitten wie erwartet ab. Die Logik funktioniert soweit und liefert dem Endbenutzer die geeignetsten Server als Resultat. Verbesserungspotenzial besteht primär noch beim Front-End, da hier noch Unklarheiten in der Funktionsweise bestehen. Des Weiteren ist es noch nicht optimal, dass die Konfigurationsdatei auf dem Filesystem selbst angepasst werden muss und nicht über das Dashboard gesteuert werden kann. Da dies jedoch nicht PA relevant ist, wird dies als Verbesserung für später vermerkt.

# Schlusswort

Durch das Erstellen dieser Logik zur Verteilung von neuen Datenbanken, sind wir der Komplett-Automatisierung des Bestellprozesses einen kleinen Schritt nähergekommen. Bereits heute ist es durch diese Logik möglich, den Mitarbeitern des Datenbank-Teams eine der letzten repetitiven Arbeiten abzunehmen und die Effizienz im Betrieb weiter zu steigern. Da die Logik zukunftsorientiert geschrieben wurde, ist sie einfach erweiterbar und somit flexibel genug um auch den kommenden Ansprüchen gerecht zu werden.

Mir persönlich hat diese Arbeit viel Spass bereitet. Einen Arbeitsauftrag von A-Z in diesem Umfang durchführen zu können ist nicht etwas Alltägliches und kommt zumeist in einem eher kleineren Stil vor. Vor allem bei der Planung konnte ich einige neue Erkenntnisse gewinnen. Die filigranere Aufteilung der Arbeitspakete zeigte mir auf, wo ich etwas mehr Zeit einplanen sollte und wo nicht. Was beispielsweise bei der Realisierung des Objekts oder bei der Soll-/Ist-Auswertung der Fall war. Nichtsdestotrotz konnte ich den Steinen, die mir bei der Datenbankauthentifizierung in den Weg gelegt wurden, ausweichen und somit noch neue Erfahrungen für die Zukunft sammeln. Für ein anderes Mal nehme ich mir noch mit, Abgleiche mit dem Auftraggeber oder den Fachspezialisten etwas weniger oft, dafür intensiver anzusetzen.

# Abkürzungsverzeichnis

|  |  |
| --- | --- |
| **Abkürzung** | **Bedeutung** |
|  |  |

Tabelle 8: Abkürzungsverzeichnis

# Glossar

**Keine Indexeinträge gefunden.**

# Abbildungsverzeichnis

[Abbildung 1: NIM-Spielansicht 7](#_Toc61801847)

[Abbildung 2: UML-Aktivitätsdiagramm 8](#_Toc61801848)

[Abbildung 3: Flussdiagramm 9](#_Toc61801849)

[Abbildung 4: Test Spielansicht 12](#_Toc61801850)

# Tabellenverzeichnis

[Tabelle 1: Änderungsverzeichnis 2](#_Toc61801851)

[Tabelle 2: NIM-Funktionen 7](#_Toc61801852)

[Tabelle 3: NIM-Spiel Array 7](#_Toc61801853)

[Tabelle 8: Abkürzungsverzeichnis 17](#_Toc61801854)

# Codeverzeichnis

[Code 1: Header-Bibliotheken 5](#_Toc61801855)

[Code 2: Funktion Scanf 5](#_Toc61801856)

[Code 3: Funktion CreateRandomNimNum 10](#_Toc61801857)

[Code 4: Funktion ShowNimGame 10](#_Toc61801858)

[Code 5: Funktion NewMove 11](#_Toc61801859)

[Code 6: Funktion AskPlayer 11](#_Toc61801860)

[Code 7: Funktion CheckGameStatus 11](#_Toc61801861)

# Quellenverzeichnis

|  |  |
| --- | --- |
| [1] | R. Laich, „NIM-Spiel,“ 2021. |

# Anhang

1. Mindmap
2. Auftragsübersicht
3. Systemübersicht Back-End
4. Projektjournal
5. Abnahmeprotokoll
6. Betriebshandbuch
7. Quellcode
8. Logdateien