|  |
| --- |
| 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

20.01.2021

Analyse NIM-Spiel

HFIE 2021 – Programmierauftrag

Änderungsverzeichnis

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Datum** | **Status** | **Autor** | **Kommentar** |
| 0.1 | 17.01.2021 | Entwurf | Ken Häsler | Entwurf |
| 1.0 | 20.01.2021 | Überarbeitung | Ken Häsler | Freigabe |

Tabelle : 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](#_Toc62059245)

[1 Aufgabenstellung 5](#_Toc62059246)

[1.1 Ausgangslage 5](#_Toc62059247)

[1.2 Detaillierte Aufgabenstellung 5](#_Toc62059248)

[1.2.1 Anforderungen 5](#_Toc62059249)

[1.3 Vorgaben 5](#_Toc62059250)

[1.4 User Input 5](#_Toc62059251)

[1.5 Vorgehen 6](#_Toc62059252)

[1.5.1 Analyse 6](#_Toc62059253)

[1.5.2 Implementierung 6](#_Toc62059254)

[1.5.3 Source Code Verwaltung und Hilfestellungen 6](#_Toc62059255)

[1.6 Bewertung 6](#_Toc62059256)

[Teil 2](#_Toc62059257)

[1 Analyse 7](#_Toc62059258)

[1.1 Grundüberlegungen 7](#_Toc62059259)

[1.2 Speicherung und Repository 7](#_Toc62059260)

[2 UML-Diagramm 8](#_Toc62059261)

[3 Flussdiagramm 9](#_Toc62059262)

[4 Funktionen 10](#_Toc62059263)

[4.1 CreateRandomNimNum 10](#_Toc62059264)

[4.2 ShowNimGame 10](#_Toc62059265)

[4.3 NewMove 11](#_Toc62059266)

[4.4 AskPlayer 11](#_Toc62059267)

[4.5 CheckGameStatus 12](#_Toc62059268)

[5 Testing 13](#_Toc62059269)

[5.1 Vorarbeiten 13](#_Toc62059270)

[5.2 ShowNimGame 13](#_Toc62059271)

[5.2.1 Testmotiv 13](#_Toc62059272)

[5.2.2 Mittel und Methoden 13](#_Toc62059273)

[5.2.3 Erwartungen 13](#_Toc62059274)

[5.2.4 Resultat 13](#_Toc62059275)

[5.3 NewMove 14](#_Toc62059276)

[5.3.1 Testmotiv 14](#_Toc62059277)

[5.3.2 Mittel und Methoden 14](#_Toc62059278)

[5.3.3 Erwartungen 14](#_Toc62059279)

[5.3.4 Resultat 14](#_Toc62059280)

[5.4 AskUser 15](#_Toc62059281)

[5.4.1 Testmotiv 15](#_Toc62059282)

[5.4.2 Mittel und Methoden 15](#_Toc62059283)

[5.4.3 Erwartungen 15](#_Toc62059284)

[5.5 CheckGameStatus 15](#_Toc62059285)

[5.5.1 Testmotiv 15](#_Toc62059286)

[5.5.2 Mittel und Methoden 15](#_Toc62059287)

[5.5.3 Erwartungen 15](#_Toc62059288)

[5.5.4 Resultat 15](#_Toc62059289)

[5.6 Schlussfolgerung 16](#_Toc62059290)

[6 Schlusswort 17](#_Toc62059291)

[7 Abbildungsverzeichnis 18](#_Toc62059292)

[8 Tabellenverzeichnis 19](#_Toc62059293)

[9 Codeverzeichnis 20](#_Toc62059294)

[10 Quellenverzeichnis 21](#_Toc62059295)

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 : Header-Bibliotheken

## User Input

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



Code : 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 : 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 : NIM-Spiel Array

Folgend ein Beispiel zur Konsolen-Ansicht:

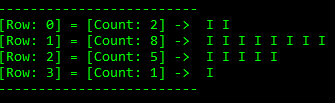


Abbildung : NIM-Spielansicht

## Speicherung und Repository

Das C-Programm wird wie von den Anforderungen verlangt auf Github gespeichert.

Link: <https://github.com/kenhaesler/HFIE2021HAKNim>​​​​​​​

Die Verwaltung des Repository führe ich durch manuelle Kommandozeilenbefehle durch.

# UML-Diagramm

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

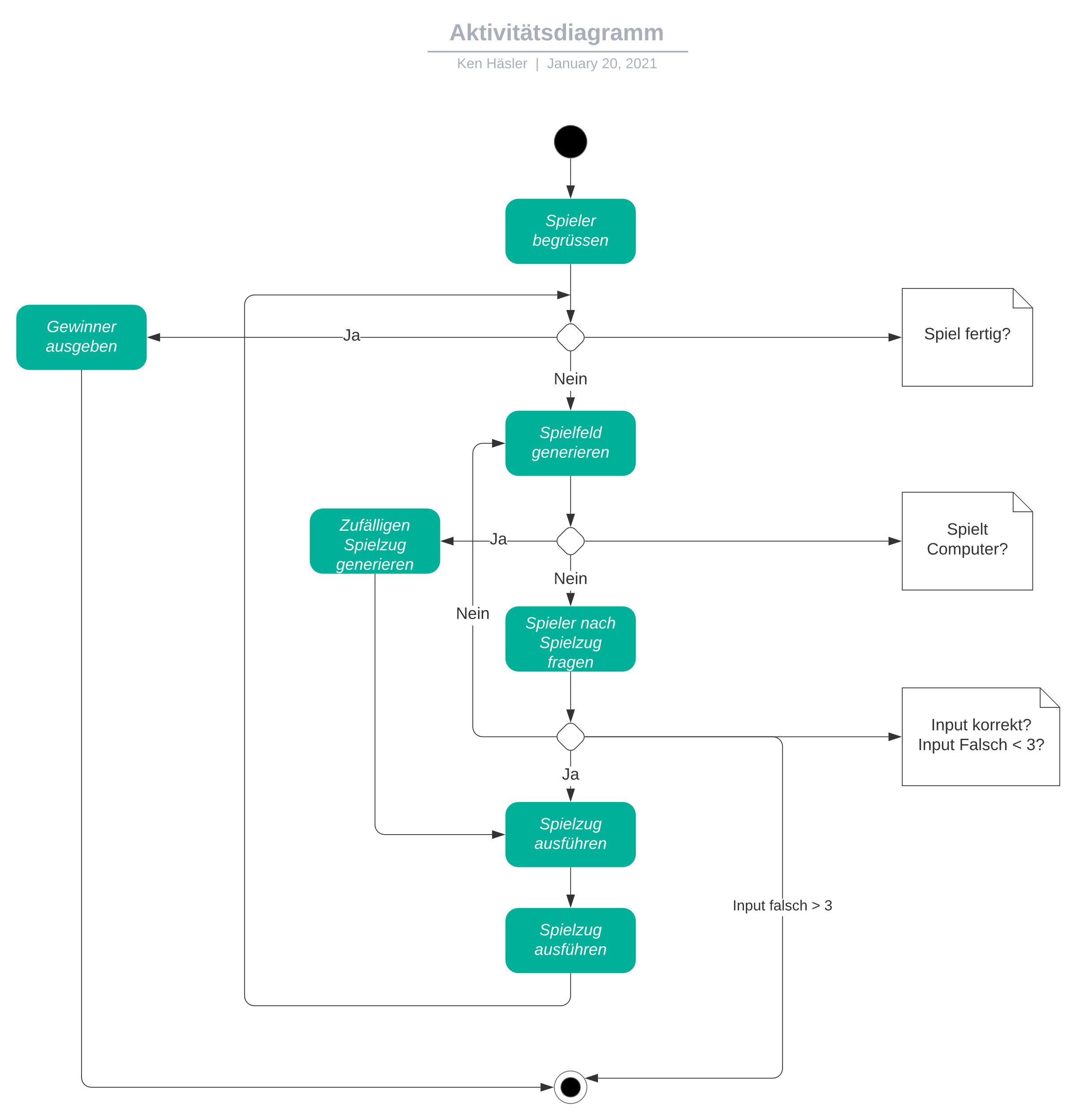


Abbildung : UML-Aktivitätsdiagramm

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

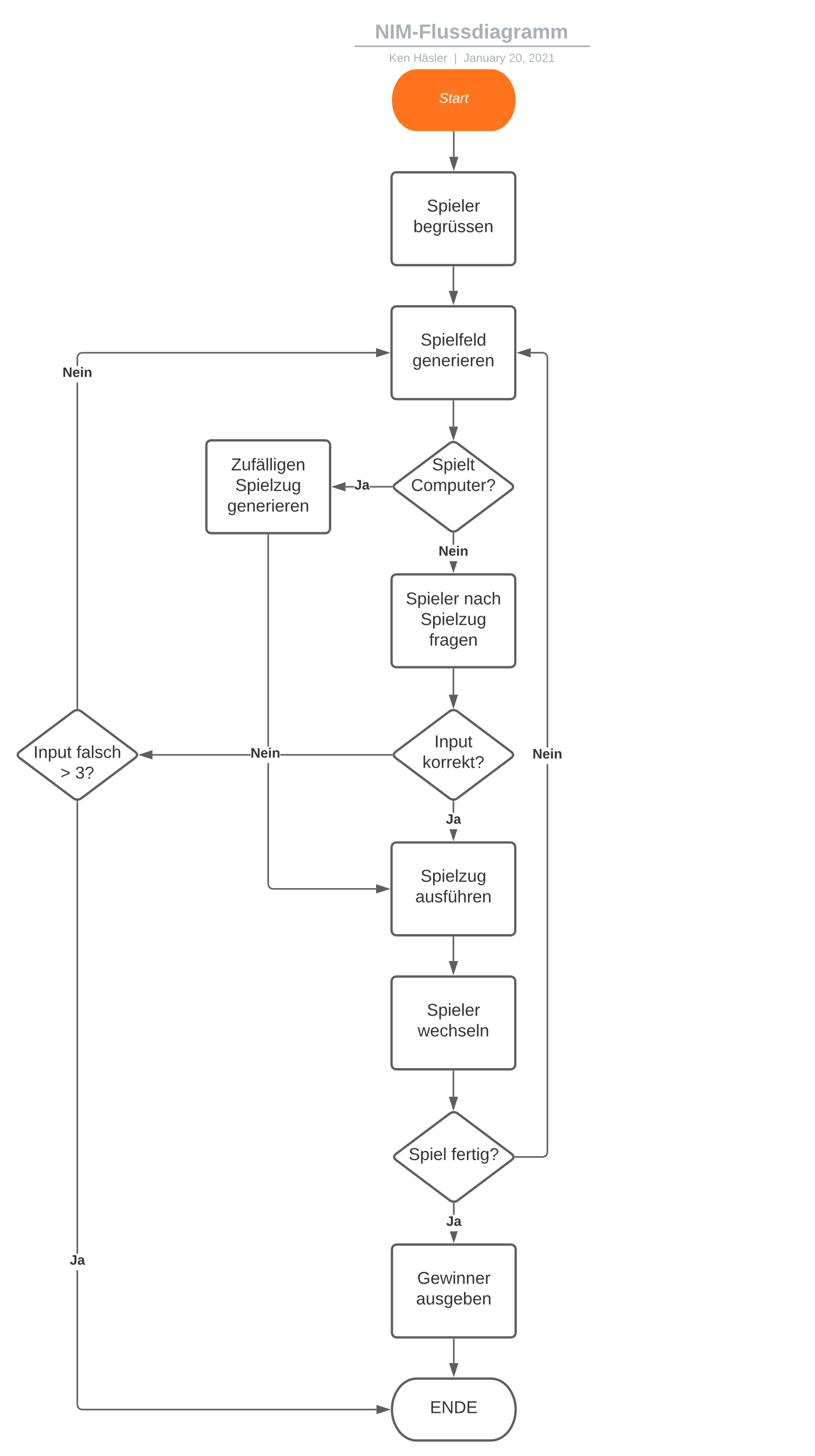


Abbildung : Flussdiagramm

# Funktionen

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

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

## CreateRandomNimNum



Code : Funktion CreateRandomNimNum

## ShowNimGame



Code : Funktion ShowNimGame

## NewMove



Code : Funktion NewMove

## AskPlayer



Code : Funktion AskPlayer

## CheckGameStatus



Code : 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. In Der Konsolenapplikation soll ein verständliches Spielfeld generiert werden.

### Resultat

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

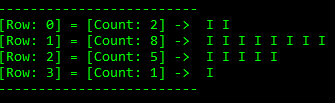


Abbildung : Test Spielansicht

## NewMove

### Testmotiv

* Kann der Spielzug für den Benutzer ausgeführt werden
* Kann der Spielzug für den Computer ausgeführt werden
* Werden Fehleingaben abgefangen

### 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

Folgend die Resultate zur NewMove Funktion:

**Spielzüge:**

Vor den Spielzügen.

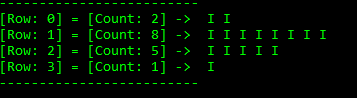


Abbildung : NewMove vor Spielzug

Nach Spielzug des Spielers (Row 0, Num 2).

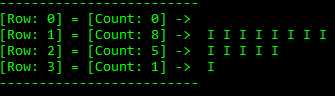


Abbildung : NewMove nach erstem Spielzug

Nach Spielzug des Computers (Row 3, Num 1).

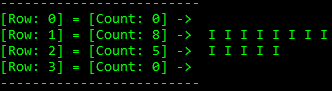


Abbildung : NewMove nach Computer-Spielzug

Fehleingabe des Benutzters.

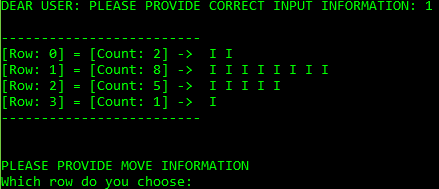


Abbildung : NewMove Fehleingabe des Benutzers

Die Spielzüge der Benutzer werden somit ausgeführt, das Spielfeld aktualisiert und Fehleingaben abgefangen. Bei einer Eingabe von Ascii-Zeichen wird das Programm sofort beendet.

## 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

Da die Funktion AskUser Teil der NewMove-Funktion ist, wurde diese durch den vorigen Test bereits getestet.

### Erwartungen

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

## CheckGameStatus

### Testmotiv

* Prüfen ob das Spiel zum richtigen Zeitpunkt abgebrochen wird

### Mittel und Methoden

Getestet wird durch das Spielen des Spiels und durch absichtliches Verlieren/Gewinnen.

### Erwartungen

Ich erwarte das das Spiel beendet wird, sobald keine Streichhölzer mehr vorhanden sind.

### Resultat

Das Spiel wird nach dem letzten Spielzug ordnungsgemäss gestoppt.

## Schlussfolgerung

Anhand der Funktionstests der verschiedenen Funktionen konnte ich feststellen, dass das Programm die grundsätzliche Funktionalität einhält. Des Weiteren konnte ich durch die Tests herausfinden, dass es durch die Eingabe einer 0, als Anzahl der zu entfernenden Hölzchen, möglich war das Spiel auszutricksen. Durch das Anpassen einer If-bedingung konnte dieser Bug dann behoben werden.

# Schlusswort

Das Programmieren dieser kleinen Arbeit hat mir beim Verständnis der Sprache C deutlich weitergeholfen. Nichtsdestotrotz konnte ich nicht alle Funktionen so umsetzen, wie ich dies initial geplant hatte. Etwas das mir immer noch Kopfzerbrechen bereitet ist das Error-Handling, welches ich von anderen «Sprachen» wie PowerShell anders gewohnt bin. Durch Tipps und Hinweise der Lehrperson konnte ich den Code generell noch etwas verbessern.

Verbesserungspotenzial sehe ich vor allem noch bei der effizienten Gestaltung des Codes, sowie bei der Herangehensweise an die Analyse, welche ich gerne zeitlich etwas hinausschiebe.

# Abbildungsverzeichnis

[Abbildung 1: NIM-Spielansicht 7](#_Toc62057211)

[Abbildung 2: UML-Aktivitätsdiagramm 8](#_Toc62057212)

[Abbildung 3: Flussdiagramm 9](#_Toc62057213)

[Abbildung 4: Test Spielansicht 14](#_Toc62057214)

[Abbildung 5: NewMove vor Spielzug 15](#_Toc62057215)

[Abbildung 6: NewMove nach erstem Spielzug 15](#_Toc62057216)

[Abbildung 7: NewMove nach Computer-Spielzug 15](#_Toc62057217)

[Abbildung 8: NewMove Fehleingabe des Benutzers 16](#_Toc62057218)

# Tabellenverzeichnis

[Tabelle 1: Änderungsverzeichnis 2](#_Toc62057219)

[Tabelle 2: NIM-Funktionen 7](#_Toc62057220)

[Tabelle 3: NIM-Spiel Array 7](#_Toc62057221)

# Codeverzeichnis

[Code 1: Header-Bibliotheken 5](#_Toc62057222)

[Code 2: Funktion Scanf 5](#_Toc62057223)

[Code 3: Funktion CreateRandomNimNum 11](#_Toc62057224)

[Code 4: Funktion ShowNimGame 11](#_Toc62057225)

[Code 5: Funktion NewMove 12](#_Toc62057226)

[Code 6: Funktion AskPlayer 12](#_Toc62057227)

[Code 7: Funktion CheckGameStatus 13](#_Toc62057228)

# Quellenverzeichnis

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