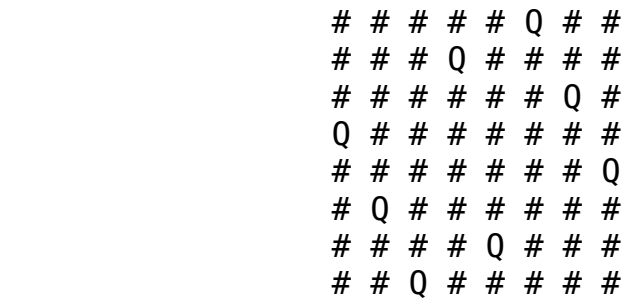
N-Damen Problem

Dokumentation



Praktikum:

Prozedurale Programmierung, Dipl. -Ing. A. Badura

Abgabe:

Paul Meffle, Mtr. Nr. 182618, 12.01.18

Inhaltsverzeichnis

1. Einleitung

In der gestellten Aufgabe geht es darum einen Algorithmus zu entwickeln um das sogenannte N-Damen Problem für N gleich drei bis einschließlich zwölf zu lösen. Das N-Damen Problem, eine schachmathematische Aufgabe, wurde im Jahr 1848 in einer Berliner Schachzeitung formuliert. Es geht darum auf einem Schachbrett mit N Reihen und Spalten N Damen so zu platzieren dass sie als “unabhängig”, sprich von keiner anderen Dame bedroht, bezeichnet werden können. Für ein normales Schachbrett mit acht Reihen und Spalten gibt es 92 verschiedene Möglichkeiten acht Damen so zu platzieren dass sie sich nicht gegenseitig bedrohen. Teil der Aufgabe ist es die Anzahl der möglichen Lösungen zu bestimmen und jede einzelne Lösung auszugeben. Darüber hinaus kann noch die Anzahl der eindeutigen Lösungen bestimmt werden (im Falle von N gleich acht zwölf), dies wird in der Aufgabe jedoch nicht berücksichtigt. Die vordergründig größte Herausforderung ist hierbei den rekursiven Algorithmus zu programmieren, jedoch sind auch die restlichen Komponenten wie Eingabe und Ausgabe nicht zu unterschätzen.

2. Analyse

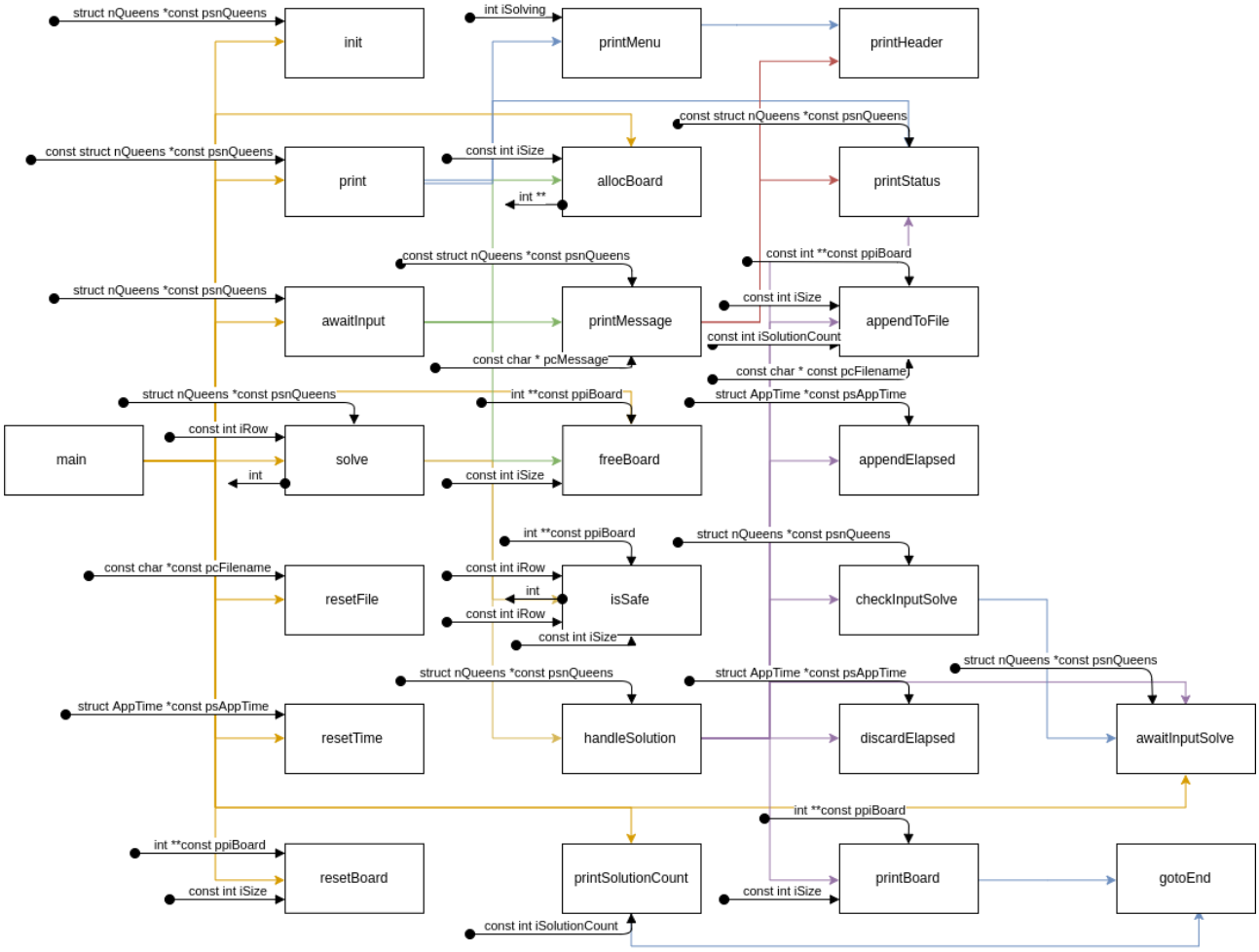
Zürst galt es sich einen groben Überblick über das Problem zu schaffen, wobei mir der englische Wikipedia Artikel „Eight queens puzzle”[1] einen guten Einstieg bat. Da dort jedoch nur ein iterativer Algorithmus beschrieben wird, weitete ich meine Recherche auf andere Seiten aus. Als Quelle für meine Implementierung des Algorithmus dienten letzendlich die Artikel „Backtracking | Set 3 (N Queen Problem)”[3] und „N-queens problem”[2]. Der Algotrithmus selbst besteht aus der rekursiven Funktion „solve” und der Hilfsfunktion „isSafe” die prüft ob die Platzierung einer Dame auf dem gegebenen Feld möglich ist.

Nachdem der Algorithmus erlfolgreich implementiert war, galt es sich Gedanken über die Struktur des restlichen Programms zu machen. Als Resultat entstanden die Module „input”, „print”, „alloc”, „solve”, „nqueens”, „main” und „save”. Außerdem passte ich die Datei „nQueensWiSe2014.h” nach meinen Vorstellungen an woraus jeweils ein „enum” für den Speicher-Modus und App-Modus enstanden. Außerdem machte ich kleinere Anpassungen in „struct nQueens” und „struct AppTime”. Das Modul „input” kümmert sich darum die Eingaben des Benutzers zu erkennen und darauf zu reagieren indem es die korrekten Eigenschaften in „struct nQueens” ändert. In „print” wird die Ausgabe des Bretts, des Menüs und der Statusleiste festgelegt. „alloc” ist für das Allokieren und Deallokieren des Bretts zuständig und „solve” enthält den Algorithmus zum lösen des N-Damen Problems. Das Modul „nqueens” ist für die initialisierung von „struct nQueens” und das zurücksetzen des Bretts. Das Modul „save” wird in eine Dll ausgelagert und enthält die Speicher-Funktionalität. Das Modul „main” legt den groben Ablauf des Programmes fest und kümmert sich um die Haupschleife.

[1] https://en.wikipedia.org/wiki/Eight_queens_puzzle (18:56 09.01.2018)
[2] https://rosettacode.org/wiki/N-queens_problem#C (18:56 09.01.2018)
[3] https://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem (18:56 09.01.2018)

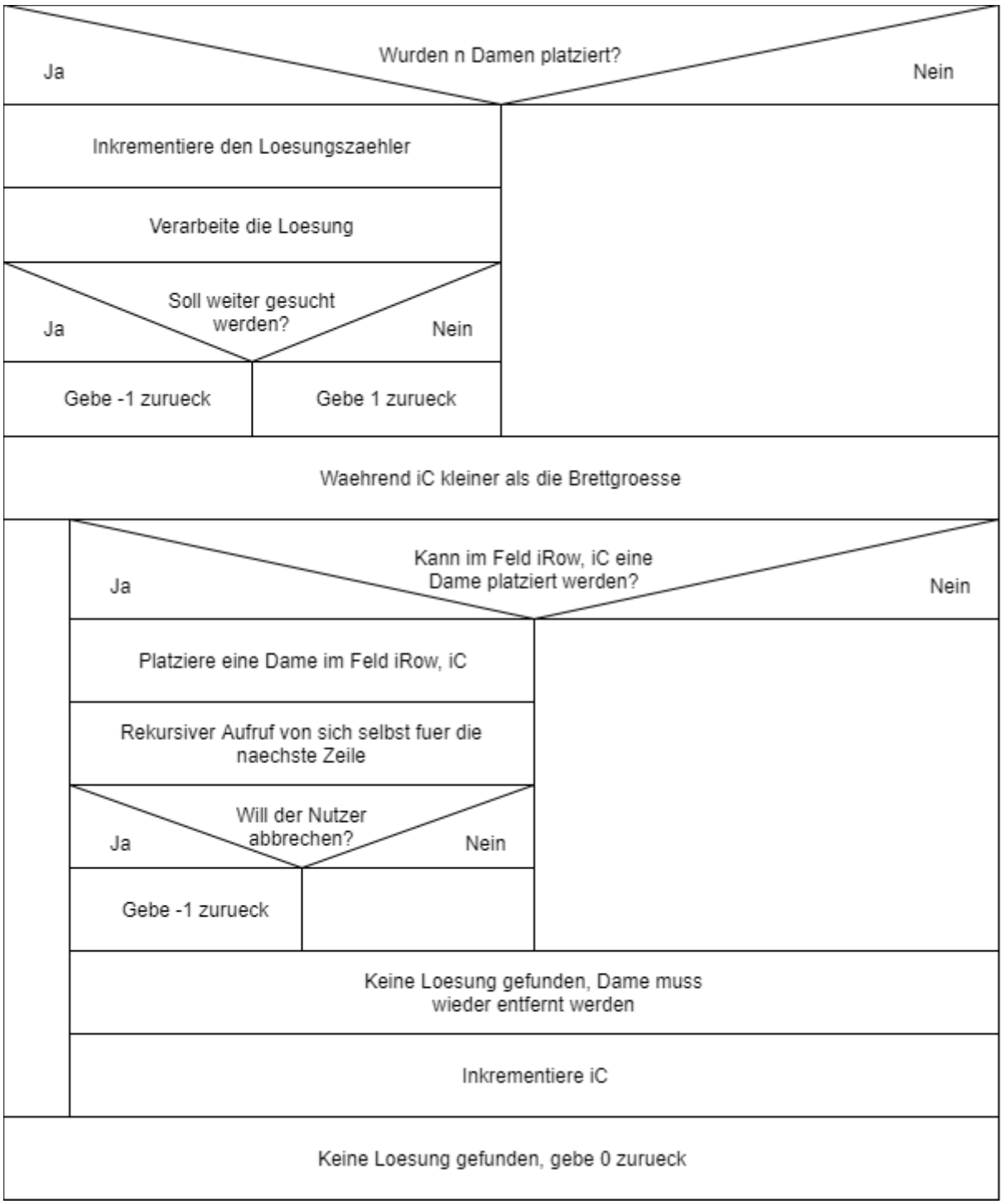
3. Design

3.1 Strukturdiagramm



Wie im Strukturdiagramm ersichtlich spielt sich der Großteil der Ablauflogik im Modul „main“ ab. Wo möglich versuchte ich statt der ganzen „nQueens“ Struktur einzelne Parameter zu übergeben um unnötige Abhängigkeiten zu vermeiden. Außerdem versuchte ich die einzelnen Aufgabenbereiche (Eingabe, Ausgabe, Speichermanagement, Problemlösung) möglichst zu kapseln, was jedoch nicht immer gelang. So werden zum Beispiel von „handleSolution“ aus Funktionen der Bereiche Eingabe, Ausgabe und Zeiterfassung aufgerufen. Es wäre schön gewesen die Lösung des Problems, Ausgabe und die Eingabe komplett voneinander zu trennen, aber auf Grund der Anforderungen entschied ich mich das in den Hintergrund zu stellen. Ein weiteres Ziel war für jede eigenständige Funktionalität eine eigene Funktion zu verwenden um so die Wiederverwendbarkeit zu steigern.

3.2 Nassi-Shneiderman Diagramm



Der Algorithmus ist wie gefordert von rekursiver Natur und basiert auf der Implementierung von „Backtracking | Set 3 (N Queen Problem)"[3]. Meine Version versucht jedoch in jeder Zeile eine Dame zu platzierung und nicht in jeder Spalte. Im Schritt „Verarbeite die Lösung" wird neben der Ausgabe der Lösung, falls im „Step" Modus, dem Speichern der Lösung, auch die Statusleiste geupdatet, die Laufzeit gemessen und überprüft ob der Benutzer das Programm beenden oder den Algorithmus abbrechen will.

Um zu entscheiden ob eine Dame in einem gegebenen Feld platziert werden kann, wird die Funktion „isSafe“ verwendet. Sie prüft ob sich über, diagonal rechts oberhalb oder diagonal links oberhalb schon eine Dame befindet und gibt dies in Form eines „int“ zurück. Da sich unter der zu platzierenden Dame keine bereits platzierte befinden kann werden diese Positionen ignoriert.

Um dem Benutzer das Abbrechen der Berechnung zu ermöglichen hat die Funktion „solve“ selbst einen Rückgabewert vom Typ „int“. Ist dieser gleich minus eins, so wurde vom Benutzer das Abbrechen der aktüllen Berechnung gefordert und alle rekursiven Aufrufe müssen beendet werden. Die „main“ Funktion entscheidet dann im weiteren ob der Benutzer nur die Berechnung abbrechen oder das Programm ganz verlassen wollte und reagiert entsprechend.

[1] https://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem (18:56 09.01.2018)

4. Testfälle

Der Ausgangspunkt jedes Testfalls ist ein neu gestartetes Programm mit folgenden Einstellungen:

- Brettgröße „4”
- App-Modus „Step”
- Speicher-Modus „SaveOff”
- Dateiname „default.txt”

Text in einfachen Anführungszeichen bezeichnet eine Menüoption und Text in doppelten Anführungszeichen eine Benutzer Eingabe. Text nach einem Pfeil beschreibt den (erwarteten) Zustand nach der Ausführung des Tests.

Fall #1
- ‘exit’
→ Programm beendet

Fall #2
- ‘begin’
→ Programm löst Problem für N gleich 4 im „Step” Modus

Fall #3
- ‘save mode’
→ Speicher-Modus wird umgeschalten, „SaveOn”

Fall #4
- ‘save mode’
- ‘begin’
→ Programm löst Problem im „Step” Modus, Lösungen werden in „default.txt” geschrieben

Fall #5
- ‘save mode’
- ‘save mode’
→ Speicher-Modus wird zweimal umgeschalten, „SaveOff”

Fall #6
- ‘app mode’
→ App-Modus wird umgeschalten, „Continuous”

Fall #7
- ‘app mode’
- ‘begin’
→ Programm löst Problem für N gleich 4 im „Continuous” Modus

Fall #8
- ‘app mode’
- ‘app mode’
→ App-Modus wird zweimal umgeschalten, „Step”

Fall #9
- ‘app mode’
- ‘save mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, Lösungen werden in „default.txt” geschrieben

Fall #10
- ‘change file’
- „test.txt”
→ Programm zeigt „filename: test.txt” in der Statusleiste an

Fall #11
- ‘save mode’
- ‘change file’
- „test.txt”
→ Programm löst Problem im „Step” Modus, Lösungen werden in „test.txt” geschrieben

Fall #12
- ‘board size’
- „6”
→ Programm zeigt „board size: 6” in der Statuszeile an

Fall #13
- ‘board size’
- „7”
→ Programm löst Problem im „Step” Modus für N gleich sieben

Fall #14
- ‘board size’
- „7”
- ‘app mode’
- ‘save mode’
→ Programm löst Problem im „Continuous” Modus für N gleich sieben, Lösungen werden in „default.txt” geschrieben.

Fall #15
- ‘board size’
- „13”
→ Programm frägt erneut nach einer Brettgröße

Fall #16
- ‘board size’
- „test”
→ Programm frägt erneut nach einer Brettgröße

Fall #17
- ‘begin’
- ‘exit’
→ Programm beendet während Lösung im „Step” Modus

Fall #18
- ‘begin’
- ‘stop’
→ Programm stoppt Lösung im „Step” Modus

Fall #19
- ‘app mode’
- ‘begin’
- ‘exit’
→ Programm beendet während Lösung im „Continuous” Modus

Fall #20
- ‘app mode’
- ‘begin’
- ‘stop’
→ Programm stoppt Lösung im „Continuous” Modus

Fall #21
- ‘board size’
- „7”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus und misst die Laufzeit (0.020s)

Fall #22
- ‘board size’
- „7”
- ‘begin’
- *Betätigen einer Taste 39 mal*
→ Programm löst Problem im „Step” Modus und misst die Laufzeit (0.029s)

Fall #23
- ‘app mode’
- ‘begin’
- *Beliebige Taste*
- ‘begin’
→ Programm löst Problem im „Step” Modus zweimal

Fall #24
- ‘app mode’
- ‘board size’
- „12”
- ‘begin’
- ‘stop’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, stoppt und löst erneut für N gleich zwölf

Fall #25
- ‘board size’
- „11”
- ‘Begin’
- *Warten*
- *Beliebige Taste*
→ Programm gibt erste Lösung aus und wartet auf Eingabe, gewartete Zeit wird nicht zur Laufzeit addiert

Fall #26
- ‘board size’
- „4”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 2 Lösungen

Fall #27
- ‘board size’
- „5”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 10 Lösungen

Fall #28
- ‘board size’
- „6”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 4 Lösungen

Fall #29
- ‘board size’
- „7”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 40 Lösungen

Fall #30
- ‘board size’
- „8”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 92 Lösungen

Fall #31
- ‘board size’
- „9”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 352 Lösungen

Fall #32
- ‘board size’
- „10”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 724 Lösungen

Fall #33
- ‘board size’
- „11”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 2680 Lösungen

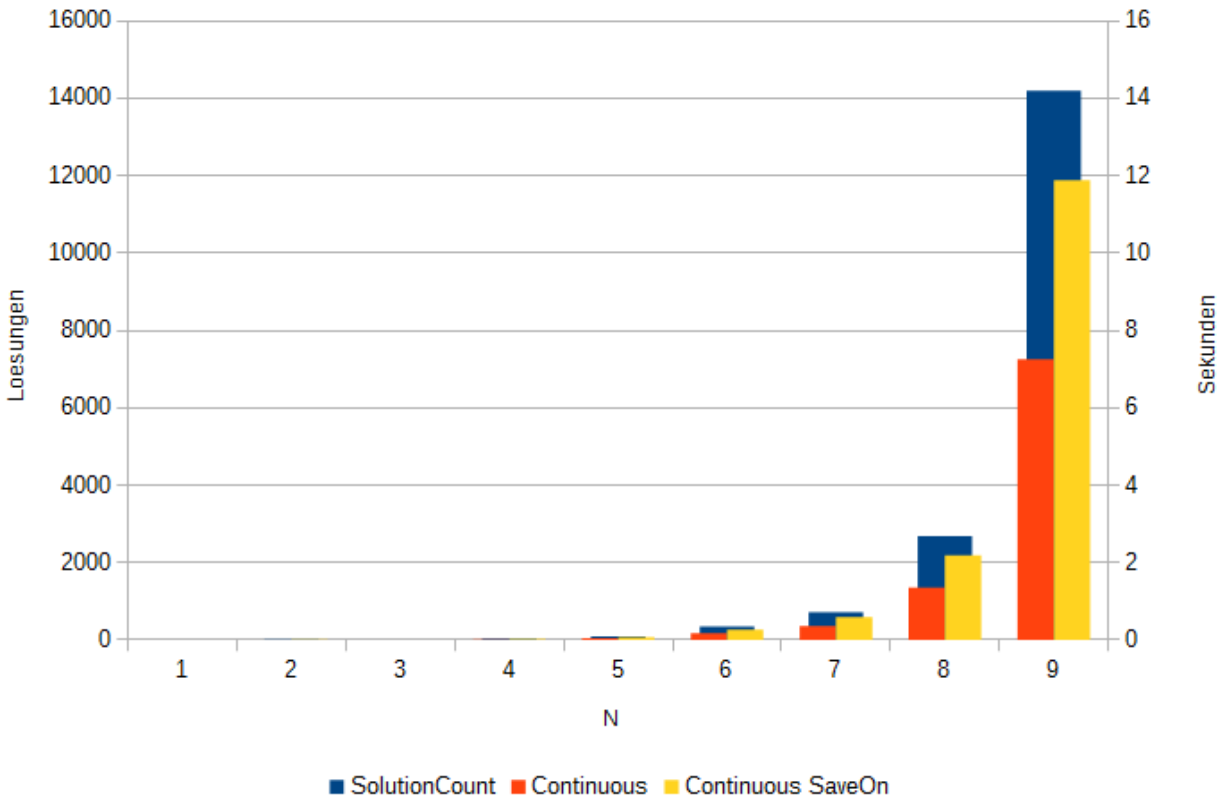
Fall #34
- ‘board size’
- „12”
- ‘app mode’
- ‘begin’
→ Programm löst Problem im „Continuous” Modus, findet 14200 Lösungen

Die Anzahl der gefundenen Lösungen für N gleich vier bis einschließlich N gleich zwölf stimmen mit den im Wikipedia Artikel „Damenproblem“[1] überein.

N	4	5	6	7	8	9	10	11	12
Anzahl	2	10	4	40	92	352	724	2680	14200

[1] https://de.wikipedia.org/wiki/Damenproblem

5. Laufzeituntersuchungen



Die Untersuchungen der Laufzeit des Programms legen Nahe dass zwischen der Größe N des Bretts und der Laufzeit eine exponentielle Beziehung besteht. Dies passt auch zu der Beziehung von N zu der Anzahl der Gefunden Lösungen, die ebenfalls von exponentieller Natur zu sein scheint. Aktiviert man die Speicherung der Lösungen nimmt die Laufzeit jeweils noch mal um Rund die Hälfte zu.

Nach einer kurzen Untersuchung mit Visual Studio scheint ein Großteil der Zeit bei der Ausgabe der Lösungen und dem updaten der Statusleiste verloren zu gehen. Derselbe Algorithmus in seiner minimalen Form, unteranderem ohne Ausgabe braucht um ein vielfaches weniger (0.606s im Vergleich zu 7.257s mit N gleich zwölf). Da die Zahl der „printf“ Statements jedoch linear Abhängig zu der Anzahl der gefundenen Lösungen sind (je ein update der Statusleiste pro Lösung im „Continuous“ Modus) kann man davon Ausgehen dass der Algorithmus auch ohne Ausgabe ein exponentielles Laufzeitverhalten hat.

6. Erfahrungen

6.1 Lernerfahrungen und Erkenntnisse

Im Verlauf des Projekts konnte einiges über die Programmiersprache C und den Umgang mit einem eigenen Projekt lernen. Es wurde mir bewusst, wie wichtig es ist sich im Vorraus Gedanken über die Struktur des Programms zu machen. Nachdem ich das Problem in mehrere kleinere Probleme unterteilt, hatte fiel es mir viel leichter diese nach und nach zu lösen. Im Umgang mit der Programmiersprache C lernte ich bei Fragen mich zuallererst an die überaus ausfürliche Onlinedokumentation von Standard-Bibliotheken und Funktionen zu wenden. Meist fand ich dort

nach kurzem lesen eine Antwort oder Lösung. Ich konnte Dinge, die ich in der Vorlesung und dem Praktikum gelernt hatte, weiter vertiefen und fühle mich nach Vollendung des Projekts sicherer im Umgang mit VisualStudio und Doxygen.

6.2 Positives und Negatives

Im Verlauf des Projekts lernte ich die im größten Teil präzisen Anforderung sehr zu schätzen, es half mir dabei den Überblick darüber zu behalten was ich schon erledigt hatte und was noch erledigt werden musste. Dagegen wünschte ich mir bei der Dokumentation etwas genauere Angaben, vor allem zu den Laufzeituntersuchungen. Im Ganzen war das Projekt eine positive Erfahrung, die mich in meinem Wissen über die Programmiersprache C und die benutzten Tools bestärkt, bzw. weitergebracht hat. Ich stimme zwar nicht mit allen Punkten des Coding Style Guides überein, dennoch finde ich es sehr wichtig dass mir zu jederzeit klar war welche Regeln ich zu beachten habe. Trotz allen Regeln gab es dennoch genügend Spielraum um das Projekt gemäß den persönlichen Vorstellungen umzusetzen.

Die Implementierung rückte mir im Vergleich zu den anderen Anforderungen etwas in den Hintergrund. Ich habe den Großteil meiner Zeit mit dem Erfüllen der Anforderungen im Bezug auf Ausgabe, Eingabe und Speicher verbracht und hätte mir einen etwas aufwendigeren Algorithmus erhofft.

Dennoch hatte ich Spaß an der Umsetzung des Projekts und konnte einige neue Erfahrungen gewinnen.