N-Damenproblem

Projektdokumentation

Autor: Marcel Grüßinger

Hochschule: HS Offenburg

Kurs: Prozedurale Programmierung

Dozent: Herr Badura

Erstelldatum: 09. Januar 2017

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Einleitung

Diese Projektarbeit setzt sich mit dem N-Damenproblem, welches eine schachmathematische Aufgabe ist, auseinander. Bei dieser Problemstellung wird die Frage aufgeworfen, wie auf einem n x n großen Schachbrett n Damen platziert werden müssen, so dass sich keine dieser nach den geltenden Schachregeln schlagen können.

Max Bezzel war die Person, die dieses Problem 1848 publik machte. Er formulierte die Aufgabe ursprünglich für ein 8x8 Schachfeld. Später verfasste Franz Nauck die Problemstellung, wie sie heute geläufig ist (sprich variabel).

In dieser Dokumentation werden unter anderem Themen, wie das Programm-Design, das Testen der Software sowie eine Laufzeituntersuchung behandelt.

Der Code des Projekts (ebenso die Dokumentation) wurde in meinem GIT-Repository, welches unter folgender URL zu erreichen ist, abgelegt:

https://github.com/Marsetex/NQueensProblem

Analysephase

Im Rahmen meiner Recherche befasste ich mich hauptsächlich mit der Art, wie der Algorithmus zur Findung der Lösungen implementiert werden kann. Dabei stieß ich auf die folgenden zwei Varianten: „Backtracking“ und „Branch and Bound“.

Backtracking verfolgt den Ansatz, Dame für Dame zu platzieren bis ein Ende erreicht ist. Dieses Ende kann eine Lösung sein (muss aber nicht). In dem zweiten Fall (welcher zu keiner Lösung führt) wird wieder Dame für Dame vom Schachbrett genommen (backtracking) bis alle Kombinationen ausprobiert wurden.

Die „Branch and Bound“-Vorgehensweise kann anhand von vier Bedingungen frühzeitig ein Ende ohne Lösung erkennen, weshalb hier im Vergleich zu der „Backtracking“-Variante Zeit gespart werden kann. Diese vier Bedingungen sind:

- Nur eine Dame pro Spalte
- Nur eine Dame pro Reihe2. No two queens share a row.
- Nur eine Dame in jeder der Diagonalen von oben rechts nach unten links
- Nur eine Dame in jeder der Diagonalen von oben links nach unten rechts

Ebenfalls bin ich durch die Recherche darauf aufmerksam geworden, dass n=27 aktuell die größte Variable ist für die alle Lösungen bekannt sind. Hier liegen insgesamt 234.907.967.154.122.528 Lösungen vor. Diese wurden erst am 19. September 2016 berechnet. Allerdings wurde dies bis jetzt nur von einem Nachfolgeprojekt nachgewiesen. Die Bestätigung des Ergebnisses durch ein zweites unabhängiges Projekt steht noch aus.

Softwaredesign

Struktogramm der Anwendung

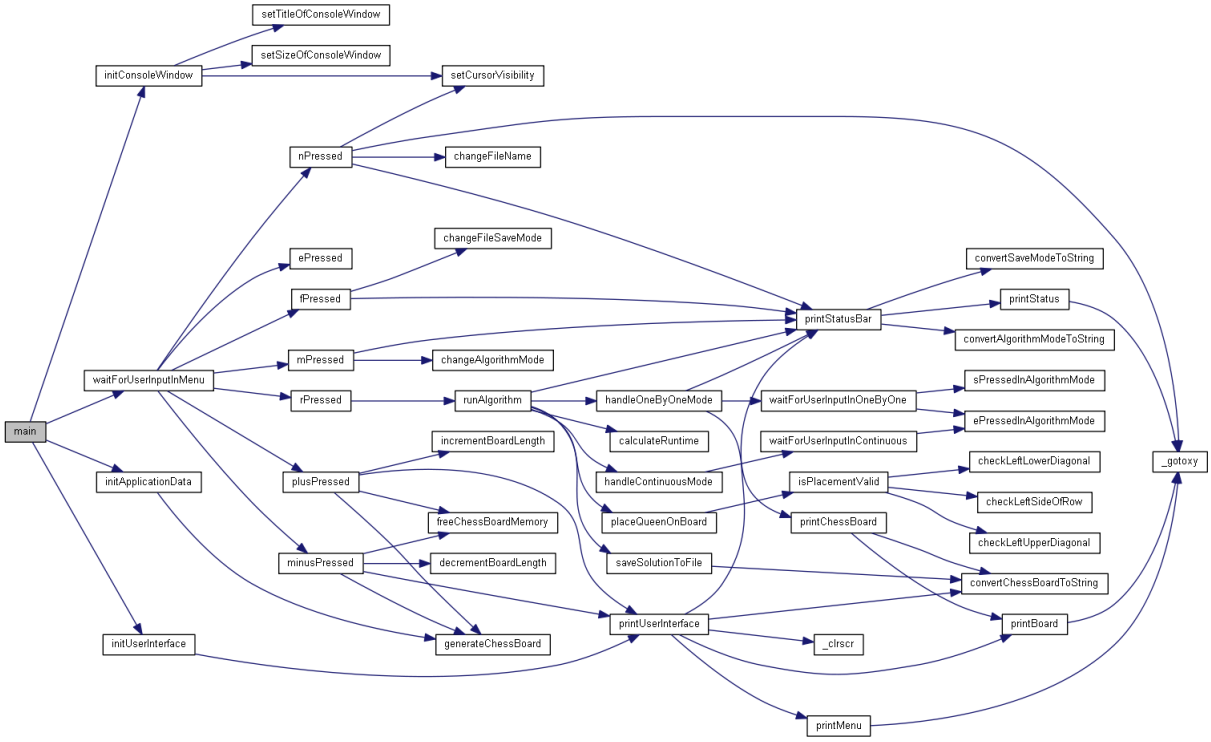

Abbildung 1: Struktogramm der Anwendung

Zu Beginn werden aus der „main“-Funktion mehrere Methoden des Initializer-Moduls aufgerufen, welches unter anderem die Standardwerte für die Elemente des ApplicationData-Structs setzt. Außerdem wird hier das Frame der Konsole initialisiert und die Benutzeroberfläche ausgegeben.

Ist die Initialisierung abgeschlossen, wird auf eine Eingabe des Anwenders im Hauptmenü der Anwendung gewartet. Dies findet in dem Modul mit dem Namen „UserInputMenu“ statt. Die eigentliche Logik für die Events (zum Beispiel, wenn die Taste „r“ gedrückt wird, um den Algorithmus zu starten) ist allerdings in „UserInputMenuLogic“ ausgelagert. Eine Implementierung, welche nur dann Gültigkeit besitzt, wenn der Algorithmus läuft, wurde für die Verarbeitung der Benutzereingaben nach dem identischen Schema umgesetzt. Hier heißen die c-Dateien „UserInputAlgorithm“ und „UsernputAlgorithmLogic“.

Innerhalb dieser Geschäftslogik-Module werden unter anderem der „DataManipulator“ und der „OutputController“ aufgerufen. Der Manipulator enthält Funktionen, um die Anwendungsdaten zu verändern, während der OutputController die Schnittstelle zu dem „UserInterface“-Modul darstellt. Der Controller bereitet die Daten zur Ausgabe vor und übergibt sie letztendlich an die Oberfläche.

Der letzte größere Bestandteil der Anwendung ist der „Algorithm“ und der „AlgorithmController“. Ersterer berechnet die Lösungen, welche dann von dem Controller-Modul weiterverarbeitet bzw. die Aufgaben weitergeleitet werden. Für diese Verarbeitung sind unter anderem die c-Dateien „FileSaveModeHandler“ und „AlgorithmModeHandler“ vorhanden, welche die entsprechenden Events behandeln, wenn einer der folgenden Modi aktiv ist: Speicherung in eine Datei, One-By-One-Algorithm oder Continuous-Algorithm.

Struktogramm der DLL

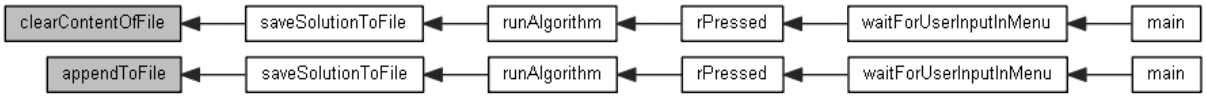

Abbildung 2: Struktogramm der DLL

Die DLL, welche ebenfalls ein Teil des Projekts ist, beinhaltet nur ein Modul (den „FileWriter“), welcher Funktionen bereitstellt, um Informationen in eine Datei zu schreiben oder diese zu leeren.

Rekursiver Algorithmus

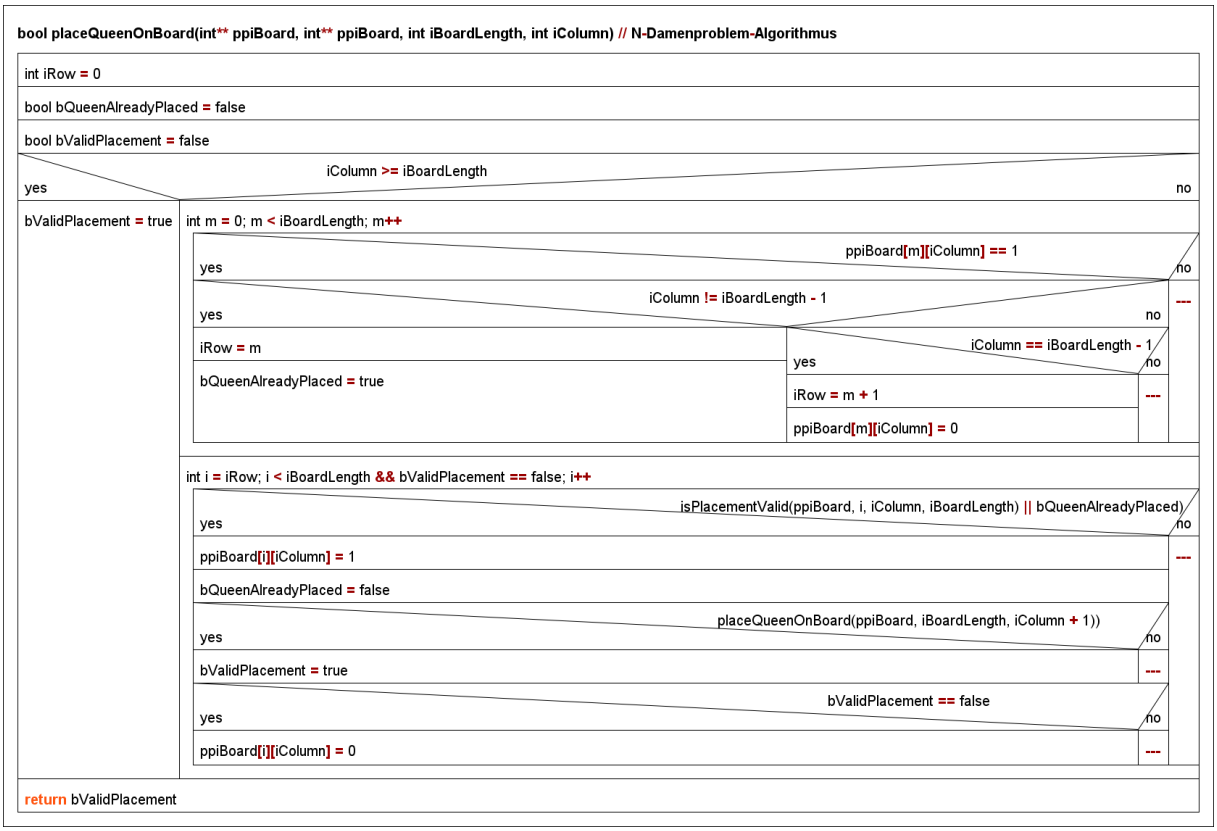

Abbildung 3: Nassi-Shneiderman-Diagramm des Algorithmus

Das Nassi-Shneiderman-Diagramm aus Abbildung 3 visualisiert den verwendeten rekursiven Algorithmus. Dieser wurde nach dem „Backtracking-Prinzip“ implementiert und versucht somit, die Damen Spalte für Spalte durch Ausprobieren zu setzen.

Bevor allerdings eine Dame auf dem Schachbrett platziert wird, prüft der Algorithmus zunächst, ob bereits eine Dame in dieser Spalte vorhanden ist. Ist dies der Fall wird die Überprüfung auf eine mögliche valide Platzierung (beschrieben in Abbildung 3) übersprungen.

Nun fährt der Algorithmus mit der nächsten Spalte fort. Hier wird wieder nach dem identischen Verfahren überprüft, ob die Dame platziert werden kann. Ist dies möglich, geht der Algorithmus in die nächste Spalte. Falls nein, springt der Algorithmus eine Spalte zurück und platziert die Dame in der vorherigen Spalte neu.

Dieses Prinzip wird solange wiederholt, bis eine Lösung gefunden wurde. Das Kriterium dafür ist, dass die Platzierung in jeder Spalte erfolgreich war.

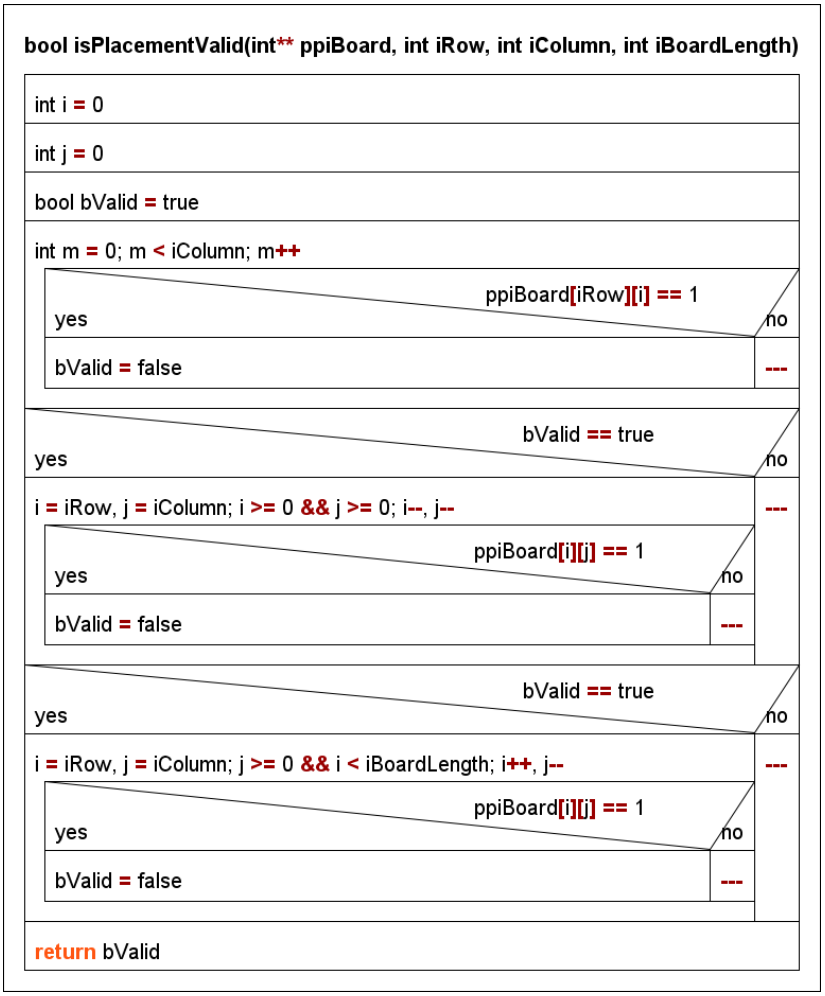

Abbildung 4: Nassi-Shneiderman-Diagramm valide Platzierung der Damen

Die Entscheidung ob eine Dame an der gewünschten Position valide gesetzt werden kann wird in Abbildung 4 dargestellt. Aus Gründe der Kompaktheit wurden die einzelnen (im nachfolgenden beschrieben) Funktionen in einem Nassi-Shneiderman-Diagram zusammengeführt. Die eigentliche Logik wurde dabei nicht verändert.

Es wird in drei Schritten überprüft ob eine Platzierung möglich ist. Zuerst wird die Zeile untersucht in der die Dame gesetzt werden soll. Dazu wird geprüft ob in einer der Spalten links von der Dame eine weitere bereits platziert wurde.

Als nächstes wird die Diagonale nach links oben (von dem gewünschten Punkt aus) auf eine zweite Dame untersucht. Schlussendlich wird ebenso die Diagonale nach links unten auf die identische Bedingung geprüft.

Findet keiner der drei Schritte eine Dame liefert diese Funktion ein „true“ an den Algorithmus zurück. Sollte jedoch eine der Phasen eine Dame finden wird sofort „false“ als Rückgabewert definiert.

Testen der Software

Spezifikation von Testfällen

Nr.	Testfall-Beschreibung	Bedingungen für Erfolg	Erfolgreich?
1	Aktivieren des One-By-One-Modus durch das Drücken der „m“-Taste	• Oberfläche gibt als Algorithmus-Modus „One-By-One“ aus	Ja
2	Aktivieren des Continuous-Modus durch das Drücken der „m“-Taste	• Oberfläche gibt als Algorithmus-Modus „Continuous“ aus	Ja
3	Schließen der Anwendung durch den Tastendruck von „e“	• Fenster bzw. Anwendung ist geschlossen	Ja
4	Ausschalten der Speicherung der Lösungen in eine Datei durch das Drücken der „f“-Taste	• Oberfläche gibt als Speicher-Modus „OFF“ aus	Ja
5	Einschalten der Speicherung der Lösungen in eine Datei durch das Drücken der „f“-Taste	• Oberfläche gibt als Speicher-Modus „ON“ aus	Ja
6	Durch das Drücken der „Plus“-Taste wird das Schachbrett korrekt vergrößert	• Schachbrett wird in der aktualisierten Größe in der Oberfläche ausgegeben • Schachbrett-Größe in der Statusbar wurde angepasst	Ja
7	Wird das 12x12-Schachbrett ein weiteres Mal inkrementiert, soll wieder von der kleinsten Größe (4x4) gestartet werden	• Korrekter Überlauf von 12x12 zu 4x4 • Das Schachbrett und die Status-Leiste wurden korrekt aktualisiert	Ja
8	Durch das Drücken der „Minus“-Taste wird das Schachbrett korrekt verkleinert	• Schachbrett wird in der aktualisierten Größe in der Oberfläche ausgegeben • Nach dem 4x4 Brett muss das 12x12 Brett folgen • Schachbrett-Größe in der Status-Bar wurde angepasst	Ja
9	Wird das 4x4-Schachbrett ein weiteres Mal dekrementiert, soll wieder bei 12x12 gestartet werden.	• Korrekter Überlauf von 4x4 zu 12x12 • Das Schachbrett und die Statusleiste wurden korrekt aktualisiert	Ja
10	Ändern des Namens (mit bzw. ohne Pfad) der Datei durch das Drücken der „n“-Taste	• Cursor erscheint bzw. verschwindet nach Betätigung der Enter-Taste • Neuer Datei-Name wird in der Oberfläche angezeigt	Ja
11	Nicht nur durch Kleinbuchstaben sollen die Funktionen (bspw. Ändern des Algorithmus-Modus, usw.) auslösen	• Großbuchstaben (d.h. zum Beispiel Shift+f) soll die Funktion auslösen	Ja
12	Die „Plus“- und „Minus“-Taste soll, identisch zu den anderen Buchstaben aus dem Menü ebenso bei gedrückter Shift- bzw. Caps lock-Taste die Funktion auslösen	• Das Vergrößern bzw. Verkleinern des Schachbretts (Ausgabe der aktuellen Werte) soll auch bei aktiver Shift-/Caps lock-Taste funktionieren	Ja
13	Cursor erscheint (unabhängig von der Schachbrett-Größe) immer unterhalb der Beschreibung „Path to file and name …“, wenn der Anwender den Datei-Namen ändern möchte	• Der Cursor erscheint am beschriebenen Ort bei Schachbrettern der Größe ab 4x4 bis 12x12	Ja

14	Ausgabe einer Fehlermeldung, wenn der Benutzer eine schreibgeschützte Datei zur Speicherung der Lösungen verwenden möchte	• Ausgabe einer Fehler-Meldung in der Status-Bar	Ja
15	Ausgabe einer Fehlermeldung, wenn der Benutzer eine Datei innerhalb eines nicht existenten Verzeichnisses anlegen möchte	• Ausgabe einer Fehler-Meldung in der Status-Bar	Ja
16	Der Dateiname (mit oder ohne Pfad) darf nur 75 Zeichen lang sein	• Alle Zeichen ab dem 76. werden abgeschnitten und nicht in der Oberfläche angezeigt	Ja
17	Unabhängig von dem Algorithmus-Modus ist der Inhalt der Datei vor dem Hinzufügen der ersten Lösung durch den aktiven Algorithmus zu leeren (Speichermodus muss aktiv sein)	• Die Datei enthält nur die Lösungen des aktuellen Schachbretts	Ja
18	Ist die Speicherfunktion vor dem Start des Algorithmus aktiviert, sollen die Lösungen in die gewünschte Datei geschrieben werden	• Alle Lösungen sind in der Datei	Ja
19	Findet der Algorithmus (unabhängig des Modus) eine Lösung, soll die aktuelle Anzahl dieser sowie die Laufzeit ausgegeben werden	• Die Status-Bar gibt unter „Amount of Solutions“ und „Runtime” die aktuellen Werte aus	Ja
20	Wird die „r“-Taste betätigt während der „Continuous“-Modus eingestellt ist, soll der Algorithmus automatisch alle Lösungen berechnen, ohne diese auf dem Schachbrett anzuzeigen	• Das Schachbrett bleibt leer • Die Statusleiste wird bei jeder gefundenen Lösung aktualisiert	Ja
21	Der laufende Algorithmus kann in dem „Continuous“- und dem „One-By-One“-Modus durch Drücken der „e“-Taste (irrelevant ob groß oder kleingeschrieben) gestoppt werden	• Fenster bzw. Anwendung ist geschlossen	Ja
22	In dem „Continuous“-Modus soll der Algorithmus nicht mit „s“ abgebrochen werden können (Algorithmus muss aktiv sein)	• Nach dem Tastendruck läuft der Algorithmus weiter	Ja
23	Die Größe des Schachfelds ist nicht veränderbar, während der Algorithmus im automatischen Modus läuft	• Weder „+“ noch „-“ haben Auswirkungen auf das Schachfeld	Ja
24	Der Algorithmus-Modus darf, unterdessen der Algorithmus aktiv ist, nicht durch Tastendruck verändert werden können	• „m“ zeigt keine Wirkung an der Oberfläche bzw. dem Algorithmus	Ja
25	Änderungen am Dateinamen sind während der Algorithmus läuft nicht vorgesehen	• Tastendruck von „n“ beeinflusst die Anwendung nicht	Ja
26	Ist der Algorithmus aktiv, kann der Speicher-Modus nicht aktiviert bzw. deaktiviert werden	• Durch das Drücken von „f“ arbeitet die Applikation ohne Änderung weiter	Ja
27	Wird die „r“-Taste betätigt, wenn der „One-By-One“-Modus eingestellt ist, soll der Algorithmus eine Lösung berechnen und diese in der Oberfläche darstellen	• Lösung wird in dem Schachbrett des User-Interfaces angezeigt • Die Statusleiste wird aktualisiert	Ja
28	Wurde eine Lösung in dem „One-By-One“-Modus gefunden, muss der Algorithmus auf einen beliebigen Tastendruck (ausgenommen „e“ und „s“) des Benutzers warten bis die nächste Lösung berechnet werden kann	• Nächste Lösung (sofern vorhanden) wird im Schachbrett dargestellt • Status-Bar wurde aktualisiert	Ja
29	In dem „One-By-One“-Modus soll der Algorithmus mit „s“ bzw. „S“ abgebrochen werden können (Algorithmus muss aktiv sein)	• Der Programm-Status wechselt zu „Pending…“ • Fenster bzw. Anwendung wird nicht geschlossen	Ja

Tabelle 1: Testfälle

Vergleich der Anzahl der gefundenen Lösungen

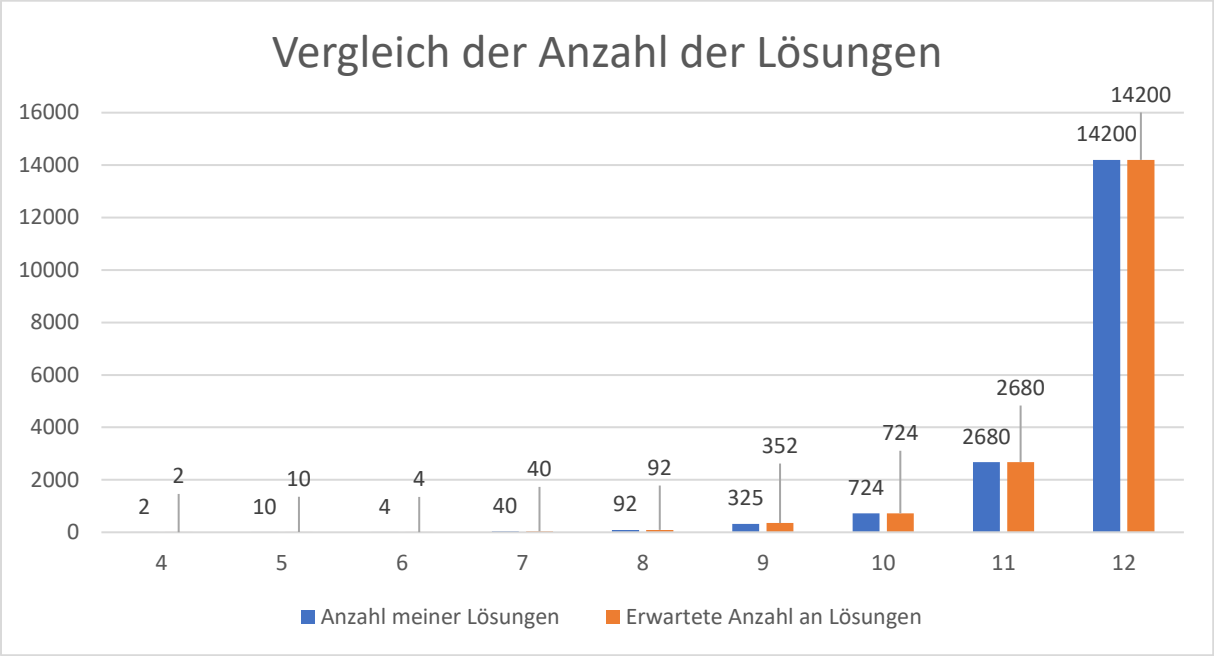

Abbildung 5: Vergleich der Anzahl der Lösungen

Der Vergleich der Lösungen zeigt, dass meine Anwendung die erwarteten Lösungen für die entsprechende Anzahl an Damen berechnet.

Grundsätzlich kann der Abbildung 5 entnommen werden, dass die Menge von Lösungen in unregelmäßigen Abständen mit anwachsender Zahl der Damen bzw. mit einem größeren Schachbrett steigt. Die Ausnahme hierbei bildet die Anzahl an Lösungen für ein Feld mit der Länge 6x6. Hier sind im Vergleich zum Vorgänger weniger Lösungen vorhanden.

Laufzeituntersuchung

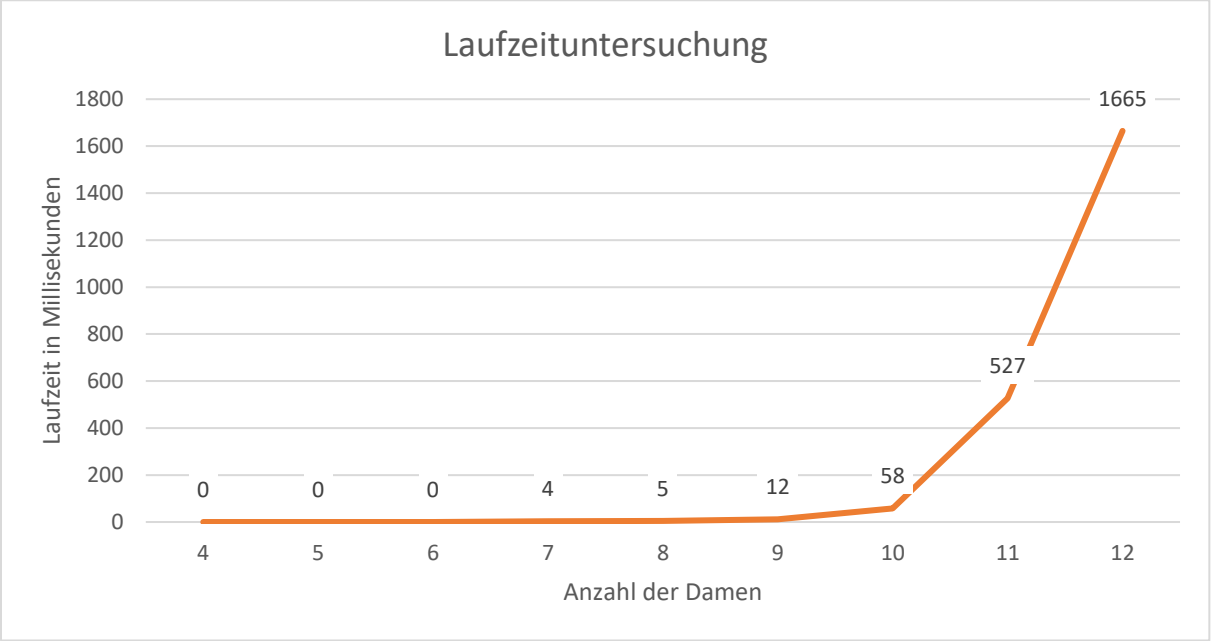

Abbildung 6: Laufzeituntersuchung

Die Abbildung 6 zeigt die Laufzeiten in Millisekunden des Algorithmus für die Menge von Damen (bzw. Größe des Schachbretts) beginnend mit der Zahl vier und endend mit der zwölf.

Aus dem Diagramm kann entnommen werden, dass für die ersten drei Werte (4, 5 ,6) die Laufzeit so gering ist, dass sie mit dem angewandten Messverfahren nicht genauer erfasst werden kann. Für die darauffolgenden Mengen 7, 8, 9 und 10 kann eine Laufzeit, welche größer als 0 Millisekunden ist, bestimmt werden. Diese Zeit steigt auch in kleinen, unregelmäßigen Schritten an.

Berechnet der Algorithmus hingegen die Lösungen für 11 und 12 Damen steigt die aufgezeichnete Laufzeit sehr stark an.

Persönliche Lernerfahrungen

Grundsätzlich lässt sich aus meiner Sicht festhalten, dass dieses Projekt mir viel Freude bereitet hat. Es war interessant, sich mit den verschiedenen Lösungsvorschlägen, auf welche ich während meiner Recherche gestoßen bin, auseinanderzusetzen.

Ebenso hat es Spaß gemacht, die Anwendung zu entwerfen und später die erworbenen Kenntnisse über die Programmiersprache „C“ zur Implementierung einzusetzen. Durch den Einsatz konnte ich mein Wissen festigen und vor allem im Bereich „Pointer“ erweitern. Auch durch die Verwendung von GIT konnte ich meine dort vorhandenen Grundkenntnisse in diesem Themenbereich ausbauen.

Als kleinen Negativpunkt könnte man aufführen, dass durch die bereits vorhandenen Programmierkenntnisse im Bereich „Java“ die Aufgabe eher eine kleine Herausforderung war.

Literaturverzeichnis

[1] http://www.geeksforgeeks.org/backtracking-set-3-n-queen-problem/, zuletzt aufgerufen 09.01.2017

[2] http://www.geeksforgeeks.org/branch-and-bound-set-4-n-queen-problem/, zuletzt aufgerufen 09.01.2017

[3] http://jsomers.com/nqueen_demo/nqueens.html, zuletzt aufgerufen 09.01.2017

[4] https://www.cs.usfca.edu/~galles/visualization/RecQueens.html, zuletzt aufgerufen 09.01.2017

[5] https://de.wikipedia.org/wiki/Damenproblem, zuletzt aufgerufen 09.01.2017

[6] http://algorithms.tutorialhorizon.com/backtracking-n-queens-problem/, zuletzt aufgerufen 09.01.2017

[7] https://www.pexels.com/de/foto/schwarz-und-weiss-spiel-schach-brettspiel-40796/, zuletzt aufgerufen 09.01.2017