Institut für Formale Methoden der Informatik
Universität Stuttgart
Universitätsstraße 38
D-70569 Stuttgart

Bachelorarbeit Nr. 76

Das Geographiespiel in der Theorie und seine Realisierung als App

David Steinhart

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Ulrich Hertrampf
Betreuer:	Prof. Dr. Ulrich Hertrampf
begonnen am:	01.06.2013
beendet am:	01.12.2013
CR-Klassifikation:	F.1.3, F.2.0

Abstract

Das sogenannte „Geographiespiel“ ist ein simples Spiel, welches dem Leser eventuell in einer Variante bekannt ist. Dabei nennen zwei Spieler abwechselnd Städtenamen, wobei diese jeweils mit dem Endbuchstaben der vorherigen Stadt beginnen müssen.

Nennt Spieler A beispielsweise „Stuttgart“, so kann Spieler B „Tübingen“ als Antwort geben. Daraufhin wären „Nürnberg“ oder „New York“ mögliche Antworten. Das Ende des Spieles ist erreicht, wenn einem der beiden Spieler keine weitere Stadt einfällt und er somit das Spiel verliert. Bereits genannte Städte dürfen nicht erneut verwendet werden. Denkbar wäre auch eine Variante, in der Personen oder Automarken genannt werden.

Beim Geographiespiel wird untersucht, welcher Spieler bei einer festgelegten Städteliste und optimaler Spielweise gewinnen wird. Das Spiel wird dahingehend erweitert, dass die Anzahl der Anfangs- und Endbuchstaben nun nicht mehr auf 26 festgelegt ist. Das Auswerten einer erweiterten Version des Spieles, die eine unbegrenzte Anzahl an Buchstaben zulässt, liegt in P-SPACE. Gibt es dagegen nur endlich vielen Buchstaben, liegt das Problem in P. Zudem ist es für sehr kleine Instanzen (nur 1 bzw. 2 Buchstaben) möglich, diese in logarithmischem Platz zu lösen.

Ziel dieser Arbeit ist es, Instanzen mit endlich vielen Buchstaben genauer zu untersuchen. Einerseits wird der Fall mit 3 bzw. 4 Buchstaben betrachtet. Andererseits wird die Möglichkeit untersucht, das Auswertungsproblem für Schaltkreise auf das Geographiespiel mit endlich vielen Buchstaben zu reduzieren und so die P-Vollständig zu zeigen.

Als praktischer Teil der Arbeit wurde das Geographiespiel als App implementiert. Dies dient der Untersuchung, mit welchem Aufwand und bis zu welchen Grad die Umsetzung realisierbar ist.

Inhalt

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Das Geographiespiel gibt es in zwei unterschiedlichen Varianten:

In der ersten Variante dürfen beliebig viele Anfangs- und Endbuchstaben verwendet werden, also mehr, als in unserem Alphabet enthalten sind. Diese Variante wurde bereits ausführlich untersucht und gilt als weitestgehend gelöst [Pap94].

Die zweite Variante erlaubt nur eine endliche Anzahl von Anfangs- und Endbuchstaben. Hier gibt es neben den bereits gelösten auch noch eine Reihe ungelöster Fragen [Die09]. Unter anderem konnte eine genaue Komplexitätsklassenzugehörigkeit bisher nicht bewiesen werden. An der Stelle setzt diese Arbeit an.

Die Implementierung einer spielbaren Umsetzung bietet sich an. Um eine einfache Verbreitung zu erlauben, wurde eine Smartphone-App entwickelt.

1.2 Ziele dieser Arbeit

Das Ziel dieser Arbeit ist es, neue Erkenntnisse über das Geographiespiel mit endlich vielen Buchstaben zu erhalten. Dies umfasst neben dem theoretischen Bereich auch einen praktischen Teil.

Im theoretischen Bereich geht es darum, die Komplexitätsklasse weiter einzugrenzen. Außerdem sollen durch den Versuch, eine Reduktion auf ein P-Vollständiges Problem durchzuführen, Hinweise für das weitere Vorgehen gefunden werden.

Eine der Leitfragen im praktischen Teil ist zu evaluieren, mit welchem technischen Aufwand reale Aufgaben gelöst werden können. Hierbei ist zu zeigen, ob ein aktuelles Smartphone die notwendigen Voraussetzungen erfüllt.

1.3 Gliederung

Im ersten Teil der Arbeit werden die relevanten Grundlagen besprochen. Diese erleichtern das Verständnis der anschließenden Untersuchung. Im theoretischen Bereich werden das Auswertungsproblem für Schaltkreise sowie eine mögliche Reduktion des Problems auf das Geographie-Spiel untersucht. Zudem wird für den Fall mit 3 Buchstaben die Lösbarkeit in logarithmischem Platz nachgewiesen.

Der praktische Teil ist in zwei Abschnitte unterteilt. Im ersten Abschnitt wird die möglichst effiziente Umsetzung eines bereits bekannten Algorithmus untersucht und mögliche

Verbesserungen vorgeschlagen. Der zweite Abschnitt befasst sich mit der Umsetzung des Geographie-Spiels auf Smartphones.

Alle Ergebnisse werden zuletzt im Fazit zusammengefasst und es wird ein Ausblick auf weitere Forschungsbereiche gegeben.

2 Grundlagen

2.1 Spielregeln

Das Geographiespiel mit endlich vielen Buchstaben wird im Folgenden mit „GEO“ abgekürzt.

Eine beliebige Instanz von GEO wird als Spielsituation bezeichnet. Eine Instanz entspricht einer Liste von Städten, die vorher festgelegte Anfangs- und Endbuchstaben besitzen. Falls dies im Voraus nicht genauer festgelegt wird, ist die Anzahl der Buchstaben und Städte konstant, kann aber beliebig groß sein. Als Spielzug oder „Zug“ wird die Nennung einer Stadt bezeichnet, welche mit dem aktuell ausgewählten Buchstaben beginnt. Der Anfangsbuchstabe der folgenden Stadt muss mit dem Endbuchstaben der vorher genannten Stadt identisch sein. Jede Stadt darf nur einmal genannt werden. Grundsätzlich wird das Spiel von Spieler 1 begonnen. Anschließend führen Spieler 1 und Spieler 2 abwechselnd Züge aus. Das Spiel endet, sobald einer der Spieler keinen Zug mehr durchführen kann.

Ist die Anzahl der Züge gerade, ist Spieler 1 als letztes am Zug und verliert das Spiel.

Ist die Anzahl der Züge dagegen ungerade, ist Spieler 2 als letztes am Zug und Spieler 1 gewinnt das Spiel.

Befindet sich Spieler 1 in einer Spielsituation, in der er gewinnen kann, so ist dies eine „Gewinnersituation“. Anderenfalls ist es eine „Verlierersituation“. Zu Beginn des Spiels wird ein Buchstabe festgelegt, mit dem Spieler 1 beginnen muss. Soweit nicht anders festgelegt, bezeichnet „k“ die Anzahl der Buchstaben und „n“ die Anzahl der zu Beginn vorhandenen Städte.

2.2 Formale Definition

Wie in 2.1 erläutert, wird bei GEO von einer endlichen Liste an Städten ausgegangen, die nur eine begrenzte Anzahl Anfangs- und Endbuchstaben besitzen.

Zur Veranschaulichung dient folgende Städteliste:

Gerolsheim, Gerlingen, Gießen, Malmsheim, München, Nürnberg

Dies sind die vorher festgelegten Buchstaben:

„G“, „M“, „N“

Alle erlaubten Buchstaben sind Teil der Menge B.

$$B = Menge\ der\ Buchstaben$$

Es ergibt sich:

$$B = \{G, M, N\}$$

Da für jede Stadt nur jeweils die Anfangs- und Endbuchstabe von Bedeutung sind, können diese auf 2-Tupel verkürzt werden. Der erste Eintrag entspricht dem Anfangsbuchstaben, der zweite dem Endbuchstaben.

$$S = \{(a, b)\ \epsilon\ B \times B\}$$

Alle Elemente dieser Menge für G, M und N sind:

$$S = \{(G, G), (G, M), (G, N), (M, G), (M, M), (M, N), (N, G), (N, M), (N, N)\}$$

Da Elemente mehrfach vorkommen können, muss die Multimenge MS über S gebildet werden. In MS dürfen Elemente beliebig oft vorkommen.

Eine Übertragung der Städteliste ergibt:

$$MS = \{(G, M), (G, N), (G, N), (M, M), (M, N), (N, G)\}$$

Ein Zug kann mit Hilfe der Elemente von MS durchgeführt werden. Ein Zug von G nach M ist möglich, wenn $(G, M)\ \epsilon\ MS$. Wird (G, M) genutzt, muss dieses 2-Tupel aus MS entfernt werden. Diese Darstellung ist jedoch nicht sonderlich gut geeignet, da sie sehr unübersichtlich ist. Zudem können 2-Tupel doppelt vorkommen.

2.3 Darstellung als Graph

Die erste alternative Darstellung ist die als gerichteter Graph G.

$$G = (V, E)$$

B wird dabei in Knoten umgewandelt.

$$V = \{a \, \epsilon \, B\}$$

S repräsentiert die Kanten zwischen diesen Knoten.

$$E = \{(a, b) \, \epsilon \, S\}$$

Der Zug von Knoten G zu Knoten M ist auch hier möglich, wenn $(G, M) \, \epsilon \, E$ und (G, M) noch nicht entfernt wurde.

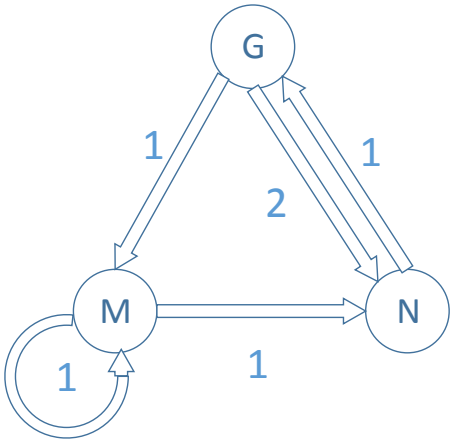

Abbildung 1: Beispiel für ein Geographie-Spiel in Graph-Darstellung

2.4 Darstellung als Matrix

Die zweite Möglichkeit ist die Darstellung als Matrix. Hierbei werden so viele Zeilen und Spalten angelegt, wie Buchstaben in B enthalten sind. Alle Einträge werden mit Null belegt. Anschließend wird für jeden 2-Tupel der Eintrag in der entsprechenden Zeile und Spalte um eins erhöht. „Gerlingen“ z.B. entspricht (G, N), darum wird in der Zeile G und der Spalte N der Eintrag erhöht. Ein Übergang von der Zeile G in die Zeile N ist dann möglich, wenn der Eintrag (G, N) größer als Null ist. Bei jedem Spielzug wird der entsprechende Eintrag verkleinert.

Beispiel für drei Buchtstaben:

$$\begin{array}{cccc} & A & B & C \\ A & a & b & c \\ B & d & e & f \\ C & g & h & i \end{array}$$

{Gerlingen, Gerolsheim, Gießen, Malmsheim, München, Nürnberg} mit den Buchstaben {G, M, N}

$$Matrix\ M = \begin{array}{c} \\ G \\ M \\ N \end{array} \begin{array}{c} \begin{array}{ccc} G & M & N \end{array} \\ \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \end{array}$$

Diese Form der Darstellung ist recht anschaulich, da hier keine doppelten Einträge mehr auftreten. Die verschiedenen Darstellungen können äquivalent ineinander umgewandelt werden.

Zu Beginn des Spiels beginnt Spieler 1 grundsätzlich in Zeile 1 bzw. mit dem entsprechenden Buchstaben. Würde Spieler 1 dagegen in einer anderen Zeile beginnen, könnte durch umschreiben eine gleichwertige Matrix erzeugt werden, in welcher sich Spieler 1 wiederum in der ersten Zeile befindet. Eine Zeile wird rot markiert, wenn Spieler 1 am Zug ist und blau, wenn Spieler 2 am Zug ist.

2.5 Gleichwertige Reduktion von Spielsituationen

Es ist möglich, viele Spielsituationen zu vereinfachen und somit die Komplexität zu reduzieren. Bei der Auswertung ergibt sich hierbei kein Unterschied [Die09]. Eine Gewinnersituation bleibt also auch nach der Reduktion eine Gewinnersituation und umgekehrt.

Als Beispiel wird wiederum die zuvor verwendete Spielsituation betrachtet, welche „Gerlingen“ und „Nürnberg“ enthält. Diese beiden Städte heben sich gegenseitig auf, da der Zug von G->N durch den Zug N->G wieder rückgängig gemacht werden kann.

Konkret bedeutet dies eine Reduktion des größeren Eintrags in der Matrix um den kleineren. Der kleinere Eintrag wird zu Null gesetzt. Dies kann für alle Einträge durchgeführt werden.

Für die Diagonaleinträge von links oben nach rechts unten entspricht diese Reduktion der Berechnung des Werts Modulo 2.

Hier ein Beispiel mit vier beliebigen Buchstaben und einer möglichen Vereinfachung.

$$\begin{pmatrix} 0 & b & c & d \\ e & 3 & g & h \\ i & j & 2 & l \\ m & n & o & 1 \end{pmatrix} => \begin{pmatrix} 0 & b-e & 0 & 0 \\ 0 & 1 & g-j & h-n \\ i-c & 0 & 0 & l-o \\ m-d & 0 & 0 & 1 \end{pmatrix}$$

Bei k Buchstaben müssen mindestens $\frac{k^2-k}{2}$ der k^2 Einträge Null sein. Das entspricht der Hälfte der Einträge, die nicht auf der Diagonalen liegen.

Zudem befinden sich alle Einträge auf der Diagonalen von links oben nach rechts unten in dem Wertebereich {0, 1}.

2.6 Lösungsalgorithmen

Es werden zwei Wege für das Auswerten von Spielsituationen gezeigt. Das Auswerten einer Spielsituation entscheidet, ob es sich um eine Gewinner- oder um eine Verlierersituation handelt.

Der erste Weg ist, mittels dynamischer Programmierung, alle möglichen Spielsituationen auszuwerten. Wurden alle Situationen mit n Städten gelöst, können alle Situationen mit (n + 1) Städten ausgewertet werden, indem auf zuvor gelöste Situationen zurückgegriffen wird. Die Anzahl der Spielsituationen verhält sich polynomiell zur Anzahl der Städte und liegt in $O\left(n^{k^2}\right)$ [Die09].

Für eine praktische Umsetzung eignet sich dieser Algorithmus jedoch nicht, da die Laufzeit bei 26 Buchstaben in $O(n^{676})$ liegt, was real nicht umsetzbar ist.

Der zweite Weg ist ein rekursiver Algorithmus. Dieser testet alle möglichen Spielverläufe und bestimmt jeweils, ob der Startspieler gewinnen kann.

Die Laufzeit des rekursiven Algorithmus und spezifische Verbesserungen werden in 4.1 genauer ausgeführt.

3 Theoretische Betrachtung

3.1 Das Auswertungsproblem für Schaltkreise

Gegeben ist eine endliche Menge boolescher Ausdrücke. Diese sind von folgendem Typ:

$$P_i := 1 \quad P_i := 0 \quad P_i := \neg P_j$$

$$P_i := P_j \vee P_k \qquad P_i := P_j \wedge P_k$$

Es muss j, k < i gelten und alle Ausdrücke dürfen nur einmal festgelegt werden.

Bekanntermaßen ist der Junktor „NAND“ funktional vollständig. Die Junktoren „Oder“, „Und“ sowie die Negation können also durch verschachteltes Aufrufen des NAND-Operators dargestellt werden. Der zu untersuchende Ansatz besteht darin, eine Spielsituation zu erstellen, deren Auswertung identisch zum wiederholten Aufruf der NAND-Funktion ist. Das Ergebnis „Gewinnersituation“ entspricht dem Wert „Wahr“ für den booleschen Ausdruck.

Als Beispiel dient der folgende Ausdruck:

$$A = \neg (B \wedge C)$$

$$B = 0$$

$$C = 1$$

Eine entsprechende Spielsituation mit Auswertung:

$$\begin{matrix} A \\ B \\ C \end{matrix}\begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} => \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} => \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\begin{matrix} A \\ B \\ C \end{matrix}\begin{pmatrix} 0 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} => \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Beginnend in Zeile 1 untersucht der rekursive Algorithmus die Zeilen 2 und 3.

Zeile 2 entspricht B und wird zu einer Verlierersituation ausgewertet.

Zeile 3 entspricht C und wird zu einer Gewinnersituation ausgewertet.

Zuletzt wertet der Algorithmus den Ausdruck $\neg (B \wedge C)$ aus und errechnet das Ergebnis „Gewinnersituation“.

Mit mehr Buchstaben lässt sich dieses Vorgehen länger fortsetzen und mehrere NAND-Funktionen können verknüpft werden. Allerdings wird irgendwann der Punkt erreicht, an

dem keine freien Zeilen mehr existieren. Würden schon belegte Zeilen neu beschrieben oder verändert werden, würde sich der gesamte Ausdruck verändern und das Ergebnis wäre nicht mehr identisch. In diesem Beispiel können B und C beide die Werte Null oder Eins annehmen. So entstehen vier verschiedene Wertekombinationen. Gibt es drei Ausdrücke, sind acht Wertekombinationen möglich. Durch Hinzufügen von linear vielen Ausdrücken lassen sich entsprechend exponentiell viele Wahrheitswerte darstellen. Bei GEO ist dies nicht möglich. Jede Spielsituation entspricht einer Wahrheitswertbelegung. Um exponentiell viele Wahrheitswerte darzustellen, werden also exponentiell viele Spielsituationen benötigt.

Um exponentiell viele Spielsituationen zu erhalten, werden wiederum exponentiell viele Städte benötigt. Bei linear vielen Städten gibt es nur polynomiell viele Spielsituationen. Es kann also gar nicht möglich sein, alle Instanzen des Auswertungsproblems für Schaltkreise so zu übertragen. Es wäre beispielsweise ein Algorithmus notwendig, bei dem sehr viele Schaltkreise auf die gleiche Spielematrix übertragen werden.

Als Alternative kann versucht werden, ein anderes P-Vollständiges Problem auf GEO zu übertragen. Der Versuch, Empty-CFG auf GEO zu reduzieren, war nicht erfolgreich. Empty-CFG bezeichnet die Frage, ob eine kontextfreie Grammatik das leere Wort enthält. Eine Reduktion wäre nach dem gleichen Schema wie zuvor denkbar. Alle Produktionsregeln können in äquivalente Zeilen in der Matrix umgewandelt werden. Auch hier ist die Menge der freien Zeilen nicht ausreichend und die Anzahl der Spielsituationen zu gering.

3.2 Lösbarkeit von Instanzen mit 3 Buchstaben in logarithmischem Platz

Die Lösbarkeit von Instanzen mit einem bzw. zwei Buchstaben in konstantem bzw. logarithmischem Platz wurde bereits bewiesen [Die09]. Nun werden Instanzen mit drei Buchstaben, aber beliebig vielen Städte betrachtet. Die entsprechende Matrix hat folgende Grundform:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

Die folgenden drei Schritte sollen die Lösbarkeit aller auftretender Spielsituationen in logarithmischem Platz zeigen.

3.2.1 Gleichwertige Reduktion der Spielematrizen

Sämtliche Spiel-Matrizen werden zu Beginn reduziert. Dies ändert das Ergebnis der Auswertung nicht. Bei der Reduktion werden b / d, c / g sowie f / h miteinander verglichen. Der größere Wert wird um den kleineren Wert reduziert, der kleinere Wert wird zu Null gesetzt. Anschließend werden a, e und i reduziert, indem der Wert Modulo 2 gesetzt wird. Jetzt sind immer mindestens drei der Werte b, c, d, f, g, h Null, wohingegen $a, e, i \in \{0, 1\}$ sind.

3.2.2 Lösen von einfachen Fällen in logarithmischem Platz

Gibt es eine Zeile, welche nicht erreicht werden kann, so kann diese ignoriert werden. Dies ist für

$$(b = 0) \wedge (h = 0) \text{ oder } (c = 0) \wedge (f = 0)$$

der Fall. Die entsprechende Zeile und die zugehörige Spalte werden aus der Matrix gestrichen. Die neue Matrix wird wie eine Instanz mit 2 Buchstaben gelöst.

Sind d / g beide Null, muss dies gesondert behandelt werden. Die Zeile 1 kann in diesem Fall nicht erneut erreicht werden, sobald sie einmal verlassen wurde.

Ist a = 0, so werden b und c überprüft. Sind b oder c größer als Null, kann die Spielsituation jeweils rekursiv für Zeile 2 oder 3 ausgewertet werden. Führt mindestens einer der Züge zum Sieg, befindet Spieler 1 sich in einer Gewinnersituation, anderenfalls nicht.

Wenn a = 1 ist, handelt es sich in jedem Fall um eine Gewinnersituation. Kann weder über b noch über c ein Sieg erreicht werden, wählt der Spieler a. Nun befindet sich Spieler 2 in

Zeile 1. Da aber hier keine gewinnbringenden Züge mehr existieren, hat Spieler 2 verloren.

3.2.3 Lösen der verbleibenden Fälle in logarithmischem Platz

Wie bereits nachgewiesen, muss jeweils eines der Pärchen b / d, c / g sowie f / h Null sein. Gleichzeitig dürfen b / h, c / f und d / g nicht gleichzeitig Null sein, da die entsprechenden Fälle sonst in den vorherigen Schritten behandelt worden wären.

b	c	f	d	g	h	
1	1	1	0	0	0	d / g
1	1	0	0	0	1	d / g
1	0	1	0	1	0	–
1	0	0	0	1	1	c / f
0	1	1	1	0	0	b / h
0	1	0	1	0	1	–
0	0	1	1	1	0	b / h
0	0	0	1	1	1	c / f

Es müssen noch folgende beiden Kombinationen untersucht werden:

$$b > 0 \land f > 0 \land g > 0$$

$$c > 0 \land d > 0 \land h > 0$$

Beide Fälle sind gleichwertig. OBdA wird dieser Fall betrachtet:

$$b > 0 \quad f > 0 \quad g > 0 \quad c = 0 \quad d = 0 \quad h = 0 \quad a \in \{0, 1\} \qquad e \in \{0, 1\} \qquad i \in \{0, 1\}$$

Übrig bleiben noch alle Spielsituationen mit den gegebenen Bedingungen. Für diese muss ebenfalls in logarithmischem Platz entschieden werden können, ob es sich um eine Gewinner- oder eine Verlierersituation handelt.

Hierfür sollen die drei unten stehenden Thesen belegt werden:

1. Jede Instanz mit $(a = 0) \land (e = 0) \land (i = 0)$ ist durch abzählen lösbar

2. Jede Instanz mit $(b > 1) \land (f > 1) \land (g > 1)$ ist durch abzählen lösbar

3. Die verbleibenden Instanzen $(b < 2) \lor (f < 2) \lor (g < 2)$ sind in konstanter Zeit lösbar

3.2.3.1 (a = 0) ∧ (e = 0) ∧ (i = 0)

Betrachtet wird eine allgemeine Spielsituation, für die a, e, i = 0 gilt.

$$\begin{pmatrix} 0 & b & 0 \\ 0 & 0 & f \\ g & 0 & 0 \end{pmatrix} => \begin{pmatrix} 0 & b-1 & 0 \\ 0 & 0 & f \\ g & 0 & 0 \end{pmatrix} => \begin{pmatrix} 0 & b-1 & 0 \\ 0 & 0 & f-1 \\ g & 0 & 0 \end{pmatrix} => \begin{pmatrix} 0 & b-1 & 0 \\ 0 & 0 & f-1 \\ g-1 & 0 & 0 \end{pmatrix}$$

Keiner der Spieler hat hier eine Wahlmöglichkeit. Sie können beide jeweils nur einen Zug wählen. Es hängt also von dem kleinsten Wert der Einträge b, f, g ab, wann ein Spieler keinen weiteren Zug durchführen kann. b, f und g werden miteinander verglichen. Der kleinste Wert bestimmt, wie oft die drei Züge wiederholt werden. Zudem müssen, abhängig vom kleinsten Eintrag, zusätzliche Züge eingerechnet werden

Hat b den kleinsten Wert, endet das Spiel in Zeile 1.

Hat f den kleinsten Wert, endet das Spiel in Zeile 2 und ein Zug wird hinzugefügt.

Hat g den kleinsten Wert, endet das Spiel in Zeile 3 und zwei Züge werden hinzugefügt.

Der kleinste der drei Werte wird mit diesen Zügen zusammengezählt. Ist das Ergebnis gerade, verliert Spieler 1, anderenfalls gewinnt er.

3.2.3.2 $(b > 1) \wedge (f > 1) \wedge (g > 1)$

Die unter 1. gezeigten Spielsituationen sind komplett durch abzählen lösbar. Das Prinzip kann auf diesen Fall übertragen werden. Wird ein Zug durchgeführt, bei dem a, e oder i von 1 auf 0 verkleinert wird, so wird dies im Folgenden als „zusätzlicher Zug“ bezeichnet.

Abhängig von a, e und i ist es möglich, bis zu drei zusätzliche Züge pro Spiel durchzuführen. Genau einer der Spieler zieht daraus einen Vorteil, wenn alle diese zusätzlichen Züge gespielt werden. Die Gesamtzahl der Züge ist anschließend entweder gerade oder ungerade. Im nächsten Schritt gilt es zu zeigen, dass es ab einer gewissen Anzahl von Städten für beide Spieler möglich ist, alle zusätzlichen Züge zu spielen. Dadurch kann für alle diese Instanzen ebenfalls durch abzählen bestimmt werden, wer das Spiel gewinnen wird.

Befindet sich Spieler 1 in Zeile 1 und es werden keine zusätzlichen Züge gespielt, befindet sich Spieler 2 nach drei Zügen in Zeile 1. Gäbe es in Zeile 1 die Möglichkeit, einen zusätzlichen Zug zu spielen, so hätten beide Spieler Gelegenheit, dies zu tun. Nach 6 Zügen befindet sich Spieler 1 wieder in Zeile 1, wobei beide Spieler jeweils einmal in jeder Zeile waren, falls kein zusätzlicher Zug durchgeführt wurde. So hatten beide Spieler die Möglichkeit, alle möglichen zusätzlichen Züge durchzuführen. Dies kann nur verhindert werden, wenn vorher bereits ein zusätzlicher Zug gespielt wird.

Ab einer gewissen Anzahl von Städten muss es also für beide Spieler immer möglich sein, alle zusätzlichen Züge zu spielen. Der Gewinner lässt sich dann bestimmen, indem

$$(\text{Anzahl der möglichen Züge}) + a + e + i$$

berechnet wird. Ist das Ergebnis gerade, handelt es sich für Spieler 1 um eine Verlierersituation, ansonsten um eine Gewinnersituation. Die Anzahl der möglichen Züge wird so bestimmt, wie es in 3.2.3.1 gezeigt wurde.

Um alle Situationen durch Abzählen lösen zu können, ist es bereits ausreichend, wenn b, f und g größer als 1 sind. Der genaue Beweis hierfür wurde durch Betrachtung aller Einzelfälle erbracht und wird hier nicht aufgeführt. Das eine solche Grenze für b, f und g existieren muss reicht allerdings, um den Beweis fortzusetzen.

3.2.3.3 (b < 2) ∨ (f < 2) ∨ (g < 2)

Die verbleibenden Fälle sind eine Teilmenge der Kombinationen, bei denen b, f, g ∈ {1, 2} und a, e, i ∈ {0, 1}. In Abbildung 2 werden einige dieser Kombinationen aufgezählt.

b, f und g werden der Reihe nach um eins erhöht. Da jeweils nur der kleinste der drei Werte einen Einfluss auf das Ergebnis hat, müssen nicht alle möglichen Kombinationen durchprobiert werden. Für a, e und i werden die möglichen Kombinationen binär aufgezählt. Der Fall, dass a, e und i Null sind wurde zuvor gesondert behandelt und muss nicht überprüft werden. Die entsprechenden Spielsituationen werden einzeln ausgewertet. Ein „V“ steht für eine Verlierersituation, ein „G“ für eine Gewinnersituation. Rot hervorgehobene Einträge stellen Ausnahmen zum abzählenden Algorithmus dar. Dieser würde hier also ein anderes Ergebnis liefern.

b	f	g	a	e	i	
1	1	1	1	0	0	V
1	1	1	0	1	0	V
1	1	1	1	1	0	G
1	1	1	0	0	1	G
1	1	1	1	0	1	G
1	1	1	0	1	1	V
1	1	1	1	1	1	G
2	1	1	1	0	0	G
2	1	1	0	1	0	G
2	1	1	1	1	0	V
2	1	1	0	0	1	G
2	1	1	1	0	1	G
2	1	1	0	1	1	V
2	1	1	1	1	1	G

Abbildung 2: Mögliche Spielsituationen mit Auswertung

Ab einer gewissen Anzahl an Städten lassen sich alle Fälle durch abzählen lösen. Entsprechend treten ab dieser Anzahl keine weiteren Ausnahmen mehr in der Tabelle auf. Die Tabelle mit Ausnahmen ist demnach endlich.

Abschließend werden alle Schritte kurz zusammengefasst. Als Eingabe wird eine beliebige Spielsituation gewählt. Nachdem diese vereinfacht wurde, wird mittels drei Fallunterscheidungen geprüft, ob die Situation rekursiv lösbar ist. Ist dies nicht der Fall, wird überprüft, ob b, f und g größer sind als Zwei oder a, e und i gleich Null sind. In diesem Fall kann das Problem durch Abzählen gelöst werden. Anderenfalls kann in einer zuvor erstellten Tabelle in konstanter Zeit das Ergebnis herausgesucht werden.

Für das Geographiespiel mit 3 Buchstaben ist es somit für jede Liste von Städten in logarithmischem Platz möglich, zu entscheiden, ob es eine Gewinnstrategie für den Startspieler gibt.

3.3 Untersuchung der Lösbarkeit von Instanzen mit 4 Buchstaben

Viele Instanzen mit 4 Buchstaben lassen sich rekursiv auf Instanzen mit 3 Buchstaben zurückführen. Dazu werden ähnliche Techniken genutzt wie die im Fall mit 3 Buchstaben. Es bleiben die folgenden vier Fälle sowie alle äquivalenten Varianten, die durch Umstellung der Buchstaben erreicht werden. Blau hervorgehobene Knoten sind mögliche Anfangssituationen, die untersucht werden müssen.

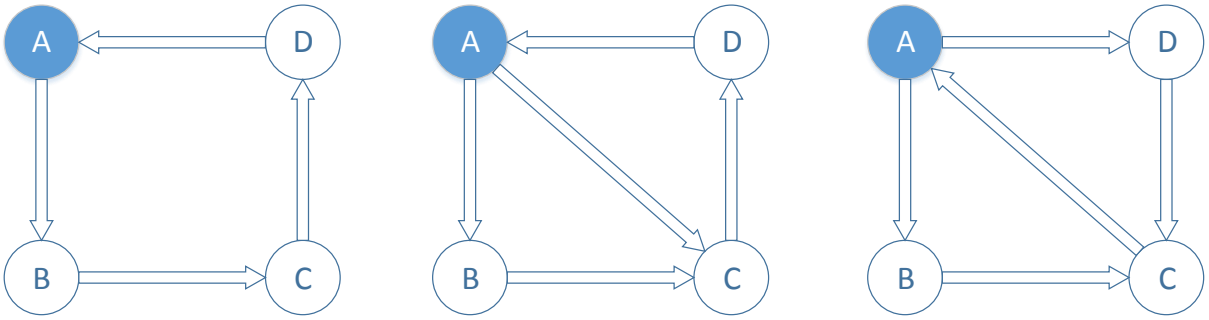

Abbildung 3: Drei Instanzen mit jeweils vier Buchstaben

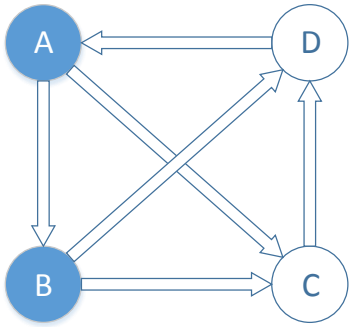

Abbildung 4: Instanz mit vier Buchstaben und allen möglichen Kanten

Die Graphen in den Abbildungen enthalten keine Schlingen. Jeder Knoten kann eine zusätzliche Schlinge besitzen. Die bisher gefundenen Methoden reichen nicht aus, um den Fall aus Abbildung 3 in logarithmischem Platz zu lösen. Allein durch Abzählen kann der Sieger nicht bestimmt werden. Durch die Schlingen entsteht eine große Anzahl an neuen Kombinationen. Es lässt sich nicht genau sagen, ob die Anzahl der so entstehenden unterschiedlichen Kombinationen in endlich viele Fälle unterteilt werden kann. Wenn keine Schlingen erlaubt sind, können alle Fälle aus Abbildung 2 mit mehreren Fallunterscheidungen ausgewertet werden. Für den Graph in Abbildung 3 konnte dies nicht gezeigt werden.

Die Untersuchung von Instanzen mit 5 Buchstaben bietet keine weiteren Anhaltspunkte. Es sind ebenfalls viele Spielsituationen enthalten, deren Auswertung in logarithmischem Platz nicht gelungen ist.

4 Praktische Umsetzung

Zu Beginn wird der Aspekt „Optimales Lösen von GEO“ im Hinblick auf die Realisierung auf einem Smartphone betrachtet. Es wird bestimmt, mit welchem Aufwand und bis zu welcher Größe verschiedene Spielsituationen ausgewertet werden können.

Anschließend wird der Aspekt „Spielbare Umsetzung“ genauer untersucht. Das Ziel war es, eine für Menschen nutzbare Applikation zu erstellen. Mit dieser soll es möglich sein, gegen einen Computergegner das Geographiespiel spielen zu können.

4.1 Optimales Lösen von GEO

Es wurde eine Android-App entwickelt, die folgende Funktionen bietet:

- Einlesen einer Liste von Städten
- Umwandeln der Städteliste in die Matrixdarstellung
- Vereinfachung der Spielematrix
- Paralleles Auswerten der Spielematrix
- Anschließende Ausgabe, welche Züge Gewinner-/Verliererzüge sind

Die Implementierung erlaubt dem Benutzer, mit einer Liste von Städten und einem festgelegten Anfangsbuchtstaben gegen den Computer zu spielen. Der Computer wertet nach jedem Zug für beide Spieler aus, ob diese noch eine Gewinnchance haben. Dem Benutzer wird angezeigt, ob er noch gewinnen kann. Auf Wunsch bekommt er alle möglichen Städte angezeigt, die noch nicht genannt wurden. Der Computer wählt zufällig einen optimalen Zug aus und nennt eine entsprechende Stadt. Der Benutzer verliert also, wenn er nicht ebenfalls optimal spielt. Diese Situation ist für den Benutzer denkbar ungünstig. Ist dem Benutzer die gesamte Liste der Städte bekannt, ist es dennoch schwierig, mehrere Züge im Voraus zu planen. Ist die Liste nicht bekannt, entscheidet ein Großteil der Benutzer sich für die erstbeste Stadt, die ihnen in den Sinn kommt.

Beide Varianten führen meistens zur Niederlage des Benutzers. Aus diesem Grund werden in 4.2 verschiedene Anpassungen vorgestellt.

4.1.1 Beschreibung des verwendeten Algorithmus

Im Folgenden wird auf den rekursiven Algorithmus eingegangen, mit dem GEO ausgewertet werden kann. Die Spielsituation muss in Matrixform vorliegen. Der Algorithmus beginnt in der Startzeile und überprüft alle Spalteneinträge in dieser Zeile. Jeder dieser Einträge entspricht einer Stadt und verweist auf eine andere Zeile der Matrix. Für jeden Eintrag, der größer als Null ist, wird ein rekursiver Aufruf durchgeführt. Der jeweilige Eintrag wird um Eins reduziert und der rekursive Algorithmus wird in der zugehörigen Zeile erneut ausgeführt.

Führt ein Eintrag zur Niederlage, wird der nächste Eintrag überprüft.

Führen alle Einträge zu einer Niederlage, wird „Niederlage“ als Antwort zurückgegeben.

Führt ein Eintrag zu einem Sieg, dann wird sofort „Sieg“ als Antwort zurückgegeben. Die verbleibenden Einträge werden nun nicht mehr berücksichtigt.

Es ist nicht optimal, wenn die Einträge in zufälliger Reihenfolge überprüft werden. Ein einziger rekursiver Aufruf mit einer positiven Antwort reicht, um die Rekursion abbrechen zu können. Daher ist es sinnvoller, zuerst Kandidaten auszuwählen, die möglichst schnell ausgewertet werden können. So kann unter Umständen schon früher die Rekursion unterbrochen werden.

Darum werden die Einträge nach der Menge der möglichen Folgezüge sortiert. Je weniger Züge es gibt, desto geringer ist die Verzweigung und entsprechend schneller kann eine Teilsituation ausgewertet werden.

Insgesamt sieht der Algorithmus in Pseudocode wie folgt aus:

Rekursiver_Algorithmus (Zeile der Matrix)

Suche alle Einträge $e_1, \ldots, e_k$ in Zeile der Matrix, die einen Wert größer 0 haben

Überprüfe für $e_1, \ldots, e_k$, in welche Zeile der Matrix sie führen und wie viele Einträge dort sind, die größer Null sind

Sortiere $e, \ldots, e$ aufsteigend danach, wie viele Einträge es jeweils gab

Führe für jeden Eintrag e, entsprechend der Sortierung, aus:

  Reduziere den Eintrag e um eins

  Rufe Rekursiver_Algorithmus auf die zu e gehörende Zeile auf

  Wenn das Ergebnis „Verlierer“ ist, gebe „Gewinner“ zurück

  Erhöhe den Eintrag e um eins

Gib „Verlierer“ zurück, sofern zuvor nicht „Gewinner“ zurückgegeben wurde

4.1.2 Auswertung

Der Algorithmus wurde in der beschriebenen Variante für ein Alphabet mit 26 Buchstaben als Android-App mit Multithreading umgesetzt.

Für bis zu 1000 Städte löst der Algorithmus eine Anfrage in weniger als einer Sekunde.

Für bis zu 1500 Städte löst der Algorithmus eine Anfrage in ca. 10 Sekunde.

Ab ca. 2000 Städten findet der Algorithmus für gewöhnlich keine Lösung. Stattdessen bricht der rekursive Algorithmus ab, da er zu tief verschachtelt ist.

Es gelten einige Ausnahmen. So gibt es auch Instanzen mit weniger als 1000 Städten, welche deutlich länger als eine Sekunde benötigen. Dies hängt hauptsächlich mit der Gleichverteilung der Städte ab. Bei gleichmäßiger Verteilung kann die Spielematrix stark reduziert werden, bei ungleichmäßiger Verteilung ist dies nicht möglich. Signifikant für die Laufzeit ist auch die Anzahl der vorhandenen Buchstaben. Je mehr verschiedene Buchstaben verwendet werden, desto länger zeigt der rekursive Algorithmus ein lineares Laufzeitverhalten. Bei weniger Buchstaben setzt schon früher eine exponentielle Laufzeit ein.

Durch eine Kombination beider Algorithmen kann dieses Problem behoben werden. Der rekursive Algorithmus speichert ab sofort alle Situation, die bereits ausgewertet wurden. Für jede neue Situation wird zuerst geprüft, ob diese schon gelöst wurde. Bei effizienter Implementierung reduziert das die exponentielle auf eine polynomielle Laufzeit. Gleichzeitig bleibt für weniger Städte das lineare Laufzeitverhalten bestehen.

4.2 Spielbare Umsetzung

Hier geht es nun um eine für den Benutzer interessante Variante.

Es wurde eine PhoneGap-App entwickelt, die folgende Funktionen bietet:

- Einlesen einer Städteliste mit zusätzlichen Informationen

- Erlaubt das Spielen gegen einen Computergegner, wobei dieser zufällig Städte wählt

- Abhängig von der Schwierigkeitsstufe spielt der Computergegner eine gewisse Anzahl an Zügen und gibt dann auf

- Der Spieler kann sich Tipps geben lassen, wenn ihm keine Stadt mehr einfällt

- Der Spieler kann aufgeben, wenn ihm mehrfach keine Stadt mehr einfällt

- Der Spieler kann sich Zusatzinfos zu Städten anzeigen lassen

4.2.1 Potenzielle Datenquellen

Bei der spielbaren Umsetzung ist die erste Aufgabe, eine möglichst vollständige Liste aller Städte der Welt zu finden. Es gibt verschiedene Quellen im Internet, mit deren Hilfe Städtelisten für einzelne Länder erstellt werden können. Zwei Beispiele hierfür sind Wikipedia [Wik] und OSM [Osm]. Beide Seiten erlauben den Zugriff auf ausführliche und aktuelle Daten. Allerdings wäre es notwendig, diese von Hand zusammen zu tragen bzw. aus einer Datenbank auszusortieren.

Die Seite Geonames [Geo] sammelt Städtedaten mit Zusatzinformationen für die gesamte Welt in einer Datenbank. Diese Daten werden von einer Community gepflegt und regelmäßig aktualisiert. Die Daten stehen kostenlos der Öffentlichkeit zur Verfügung und dürfen auch in kommerziellen Projekten eingesetzt werden. Die auf Geonames verfügbaren Daten werden verwendet, da sie am einfachsten zugänglich sind.

4.2.2 Speicherung der Daten

Diese Liste aller Städte der Welt muss nun in das Spiel eingebunden werden. Zum einem kann eine Online-Datenbank erstellt werden, mit welcher Eingaben des Spielers abgeglichen werden. Zum anderen kann die Liste direkt auf dem Endgerät abgespeichert werden. Dadurch benötigt der Spieler beim Spielen keine Internetverbindung. Allerdings steht so nur ein Teil der gesamten Liste zur Verfügung steht. Die komplette Liste ist, selbst in komprimierter Form, ca. 100 MB groß. Diese Größe ist für eine einfache Smartphone-App nicht zumutbar.

Die Liste wird in dieser Implementierung lokal gespeichert. Dadurch können alle Berechnungen direkt auf dem Endgerät durchgeführt werden und es wird kein zusätzlicher Server benötigt. Da nur ein Teil der Städte gespeichert werden kann, müssen Kriterien gefunden werden, welche Städte aufgenommen werden. Der Einfachheit wegen, werden alle Städte mit wenigen Einwohnern zuerst entfernt. Anschließend werden die Städte entfernt, bei denen die Umwandlung in das lateinische Alphabet nicht automatisiert vorgenommen werden kann. Zuletzt werden Duplikate entfernt, die entstehen können, wenn zwei oder mehr unterschiedliche Städte denselben Namen haben (z.B. Neunkirchen) oder wenn dieselbe Stadt verschiedene Bezeichnungen hat (New York / New York City). Es bleibt jeweils die Stadt mit den meisten Einwohnern erhalten.

Als Ergebnis bleiben noch ca. 50.000 Städte. Das entspricht in etwa allen Städten mit mehr als 5000 Einwohnern. Diese Grenze ist akzeptable, da die meisten verbleibenden Städte dem Spieler unbekannt und für ihn auch uninteressant wären. Die komprimierte Liste hat eine Größe von 2 MB.

Für manche Städte ist die Einwohnerzahl in der Datenbank nicht bekannt. Diese wird automatisch auf null geschätzt, wodurch in manchen Fällen mittelgroße Städte fehlen können.

4.2.3 Einheitliches Datenformat

Das Spiel soll auf Englisch erscheinen, dennoch sollen es Menschen aller Nationalitäten spielen können. Dementsprechend muss für besondere oder akzentuierte Buchstaben, die in anderen Sprachen vorkommen, eine möglichst intuitive Alternative gefunden werden. Ziel ist es, diese auf das lateinische Alphabet zu übertragen. Für den Großteil der Buchstaben gibt es festgelegte Gegenstücke. Mit einem simplen Algorithmus werden alle Einträge durchsucht und gegebenenfalls verändert. Die restlichen Einträge, für die es kein bekanntes Gegenstück gibt, werden entfernt.

4.2.4 Cross-Plattform Entwicklung

Bei der Realisierung dieses Programms wurde ein Cross-Plattform-Ansatz gewählt. Die Entwicklung muss nur einmal durchgeführt werden. Das fertige Produkt kann anschließend auf verschiedene Systeme übertragen werden, wie z.B. Android, iOS (iPhone) oder Windows Phone.

Für die Umsetzung wurde PhoneGap [Pho] gewählt. Dieses Programm erlaubt es, mit HTML5 und JavaScript zu entwickeln. Das Ergebnis entspricht einer Website, die auf jedem Smartphone dargestellt werden kann.

4.2.5 Nachteile von PhoneGap

Abgesehen von den Vorteilen gibt es auch einige Nachteile, die sich im Laufe der Entwicklung gezeigt haben.

Die Entwicklung mit PhoneGap führt auf dem Endgerät zu Performance-Einbußen. Dies stellt kein großes Problem dar, da keine aufwendigen Berechnungen durchgeführt werden. Schwierigkeiten gibt es beim Versuch, auf allen Smartphones ein einheitliches Ergebnis zu erhalten. Manche Funktionen werden von einem Smartphone-Model unterstützt, von anderen jedoch nicht. Dies führt zu abweichenden Ergebnissen auf verschiedenen Smartphones und zusätzlichem Programmieraufwand.

4.2.6 Zusätzliche Features

Zu Beginn jeder Runde wählt der Spieler eine von drei Schwierigkeitsstufen. Je schwieriger das Spiel gewählt wird, desto mehr Züge spielt der Computer. Es wird dabei zufällig entschieden, ob der Computer noch weiter spielt oder ob er aufgibt. Es ist nur eine Mindestanzahl von Zügen vorgegeben.

Fällt dem Spieler keine Stadt mehr ein, kann er sich Tipps geben lassen. Im einfachen und mittelschweren Spiel stehen ihm zwei Tipps zur Verfügung, im schweren nur einer. Es wird sofort eine Antwort für den Spieler abgegeben, wenn er einen Tipp benutzt. Dies ist relativ nützlich, da ungeübten Spielern bei schwierigen Buchstaben (wie z.B. „Q“ oder „Y“) oft keine passende Stadt einfällt. So können sie das Spiel fortsetzen und gleichzeitig neue Städtenamen kennenlernen.

Der Spieler kann sich zu Städten einige Informationen, wie z.B. Einwohnerzahl und Länderzugehörigkeit, anzeigen lassen. Interessierte Spieler können so unbekannte Städte leichter einordnen oder ihr eigenes Wissen überprüfen. Leider ist es nicht gelungen, eine Karte mit den Positionen der Städte einzubinden. Als problematisch erwiesen sich hierbei die stark abweichenden Ergebnisse auf den verschiedenen Smartphones.

Im Hintergrund und für den Spieler kaum wahrnehmbar, werden einige zusätzliche Berechnungen durchgeführt. Der Computer merkt sich, in welchen Ländern die Städte liegen, die der Spieler nennt. Er wählt dann, falls möglich, ebenfalls eine Stadt aus einem dieser Länder. Städte mit größerer Einwohnerzahl werden ebenfalls bevorzugt genannt. So werden hauptsächlich Städte genannt werden, die dem Spieler bekannt vorkommen. Das Spiel wird so für den Benutzer unterhaltsamer.

5 Zusammenfassung und Ausblick

5.1 Fazit

Die Ergebnisse der theoretischen Untersuchung können ein Hinweis dafür sein, dass ein intuitiver Ansatz nicht ausreicht, um die P-Vollständigkeit zu zeigen.

Die Umsetzung des rekursiven Algorithmus für ein gängiges Smartphone ist gelungen und entspricht den Erwartungen. Für Instanzen bis ca. 1500 Städte wird eine Auswertung in akzeptabler Zeit durchgeführt. Ein durchschnittlicher Mensch hat so fast keine Chance, gegen den Computer zu gewinnen.

Die spielbare Variante erfüllt alle geforderten Funktionen. Die zusätzlichen Informationen zu Städten erhöhen den Unterhaltungswert. Das implementierte Verlieren des Computers gibt dem Spieler eine reale Gewinnchance. Auch das Eintippen wird durch die Autovervollständigung erleichtert.

5.2 Ausblick

Im Hinblick auf eine Fortsetzung der theoretischen Untersuchung bietet sich der Vergleich mit ähnlichen Problemen an. Als möglicher Ansatzpunkt wurde die alternierende Gruppe A_5 identifiziert.

Eine Verbesserung des Auswertungsalgorithmus ist an dieser Stelle nur bedingt empfehlenswert, da es wenig Bedarf dafür gibt. Für die gezeigte Problemstellung ist der Algorithmus derzeit effizient genug.

Im Rahmen einer weiteren Arbeit könnten weitere Zusatzfunktionen und eine professionelle Spieloberfläche für die spielbare Variante erstellt werden. Das Spielvergnügen könnte damit, auch über einen längeren Zeitraum hin, erhalten bleiben.

Literaturverzeichnis

[Die09] Diekert, Volker und Hertrampf, Ulrich: *Komplexität der Geographie*. Informatik als Dialog zwischen Theorie und Anwendung: Festschrift für Volker Claus zum 65. Geburtstag, 119–132. Vieweg+Teubner, 2009.

[Geo] Geonames – Datenbank mit sämtlichen Städten der Welt. http://www.geonames.org/

[Osm] Open Street Maps – Datenbank mit kostenlosen Kartendaten. http://www.openstreetmap.de/

[Pap94] Papadimitriou, Christos: *Computational Complexity,* 460–462. Addison-Wesley, 1994.

[Pho] Phonegap – Anwendung für Cross-Plattform-Entwicklung. http://phonegap.com/

[Wik] Wikipedia – Liste der Städte in Deutschland. http://de.wikipedia.org/wiki/Liste_der_Städte_in_Deutschland

Alle URLs wurden zuletzt am 06.10.2013 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben. Wörtliche und sinngemäße Übernahmen aus anderen Quellen habe ich nach bestem Wissen und Gewissen als solche kenntlich gemacht.

Stuttgart, den 08. Oktober 2013 _____