Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelorarbeit Informatik

Komplexität von Computerspielen

Spielen unter Zeitdruck

Petra Wolf

1. Dezember 2016

Gutachter

Jun. Doz. Dr. Britta Dorn
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Betreuer

Jun. Doz. Dr. Britta Dorn
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

M.Sc. Janosch Döcker
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Wolf, Petra:
Komplexität von Computerspielen
Spielen unter Zeitdruck
Bachelorarbeit Informatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: September 2016 - Dezember 2016

Zusammenfassung

In dieser Arbeit wollen wir die Komplexität von Computerspielen betrachten. Dabei werden wir nicht einzelne konkrete Spiele untersuchen, sondern Kombinationen von Spielelementen identifizieren, deren Auftreten in einem Spiel diesem eine gewisse Komplexität verleihen. Wir bauen dabei auf der Arbeit von Giovanni Viglietta [Vig14] auf, der solche Kombinationen von Spielelementen in Metatheoremen untersucht hat. Zunächst werden wir seine Arbeit vorstellen und an dem Computerspiel „Super Mario World 2: Yoshi's Island“ anwenden, um dessen NP-Schwere zu zeigen. Dann kommen wir zum Hauptergebnis der vorliegenden Arbeit, zwei von mir entwickelten Metatheoremen, die beide das Spielelement der Zeit beinhalten. Mit diesen Theoremen werden wir zuletzt das Spiel „Super Mario Bros.“ auf seine NP-Schwere untersuchen.

Danksagung

An erster Stelle möchte ich Dr. Britta Dorn für die hervorragende Betreuung und das ausführliche Feedback danken. Ihre positiven Rückmeldungen waren für mich stets eine Motivation. Herzlichen Dank!
Des weiteren möchte ich mich bei Janosch Döcker bedanken, der mir immer eine schnelle Hilfe bei auftretenden Fragen war.
Mein Dank gilt auch Prof. Dr. Klaus-Jörn Lange, der mich mit seinen spannenden Vorlesungen für das Themengebiet der Theoretischen Informatik begeistert hat. Insbesondere möchte ich mich für die Diskussionen während der Forschungsphase bei ihm bedanken.
Nicht zuletzt möchte ich mich bei allen bedanken, die mich während dieser Arbeit unterstützt und motiviert haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

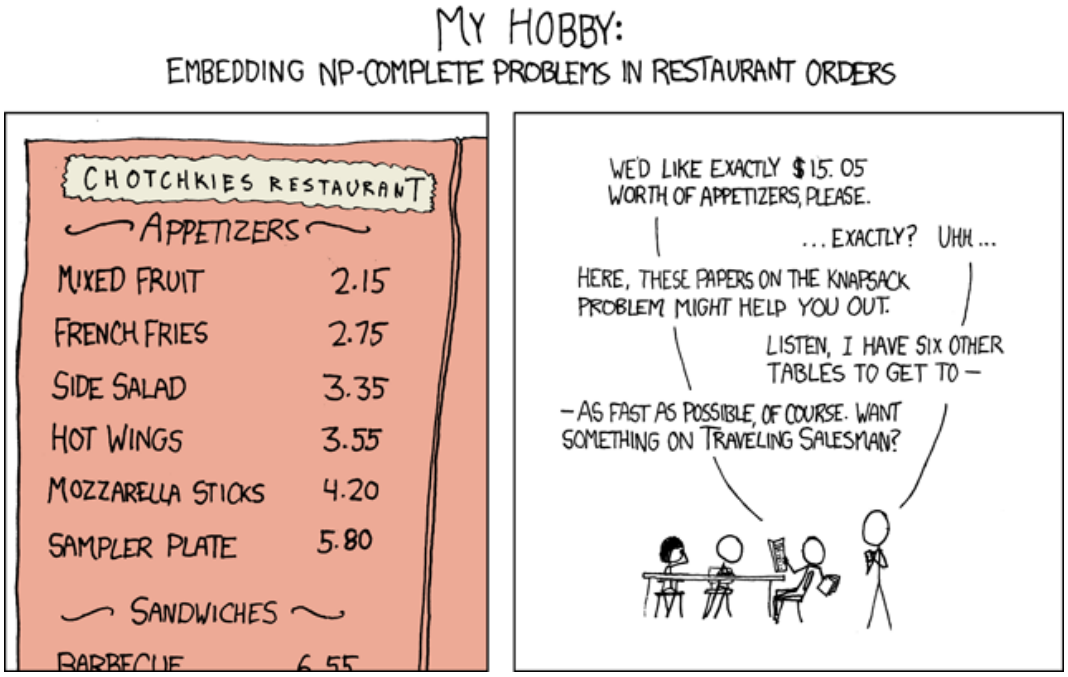

Abbildung 1.1: NP-schwere Probleme begegnen uns häufiger, als wir denken [Int07].

Der theoretischen Informatik und insbesondere der Komplexitätstheorie haftet der Ruf an, trocken und langweilig zu sein. Ihnen wird vorgeworfen, in der praktischen Welt keine Anwendung zu finden. Aber die Komplexitätstheorie begegnet uns im Alltag häufiger, als wir es vermuten, wie es unsere Einführungsgrafik (Abb. 1.1) illustriert. Beim Einkaufen im Supermarkt wollen wir zum Beispiel eine optimale Auswahl an Einkaufsgütern treffen, ohne dass deren Preis den Geldbetrag in unserem Geldbeutel übersteigt. In

der Komplexitätstheorie begegnet uns diese Aufgabenstellung als KNAPSACK Problem wieder.
Wenn sich nun auch die wenigsten Menschen in ihrem späteren Leben mit der Komplexitätstheorie und der Frage, ob P = NP ist, auseinander setzen werden, so haben sich doch die meisten von ihnen bereits in ihrer Kindheit mit NP-schweren Problemen befasst. Denn wer kennt sie nicht, die alten klassischen Nintendo Spiele der 80er und 90er Jahre. Wer hat nicht mit Mario Pilze verspeist und ist auf Schildkröten gesprungen, hat als Link Prinzessin Zelda gerettet, oder war in Pokémon auf der Jagd, sie alle zu fangen und der allergrößte Trainer zu werden?
Was, wenn ich Ihnen nun sage, dass Sie sich dabei mit NP-schweren Problemen befasst haben? In der Tat wurde von Greg Aloupis et al. [ADGV15] 2015 gezeigt, dass diese klassischen Nintendo Spiele alle mindestens NP-schwer sind. Nun, woher kommt es, dass wir Menschen uns anscheinend gerne mit besonders schweren Problemstellungen beschäftigen? Meistens wird die Klasse NP dadurch charakterisiert, dass sie Probleme enthält, für die wahrscheinlich keine effizienten Algorithmen existieren. Ich persönlich sehe das Besondere der Klasse NP darin, dass es für die Probleme keinen Hinweis auf die Lösung gibt. Es gibt kein allgemeines Verfahren, dass man stupide anwenden kann, um auf die Lösung zu kommen. Diese Eigenschaft verleiht dem Lösen solcher Probleme meiner Meinung nach eine gewisse Herausforderung. Man kann einer Probleminstanz nicht von Anfang an sofort ansehen, ob sie lösbar ist, sondern man ist gezwungen, verschiedene Ansätze auszuprobieren. Dabei kann man feststellen, dass der verwendete Ansatz falsch war und man wird zu einem Umdenken seiner Strategie gebracht. Dies entspricht der Art, wie Menschen ihre Umgebung wahrnehmen und Probleme, auf die sie treffen, lösen, viel mehr, als ein wiederholtes Anwenden eines vorgegebenen Algorithmus. Meiner Ansicht nach ist dies der Grund, warum so viele erfolgreiche Spiele mindestens NP-schwer sind.
Die Bestimmung der Komplexität von Computerspielen ist somit keine Spielerei gelangweilter Theoretiker, sondern kann uns sogar dabei helfen, das Problemlösungsverhalten von Menschen zu verstehen und künstliche Intelligenzen zu trainieren, um das gleiche Verhalten zu erlernen. Erste Schritte auf diesem Gebiet wurden schon gemacht (siehe Kapitel 3) und ich hoffe, dass noch weitere folgen werden. Die Bestimmung der Komplexität weiterer Spiele, die ich mit meinen Ergebnissen in dieser Arbeit voran bringen möchte, kann dazu beitragen, Computerspiele als Forschungsobjekte in weitere Bereiche der Natur- und Geisteswissenschaften einzubringen.

Um Spiele, bei denen es sich meistens um endliche Objekte handelt, sinnvoll komplexitätstheoretisch untersuchen zu können, wollen wir abstrahierte bzw. verallgemeinerte Versionen von Spielen betrachten. Dabei besteht die Abstrahierung meist daraus, beliebig große Level zu betrachten. Zudem wird ange-

nommen, dass das gesamte Level gleichzeitig zu sehen ist und die Zustände aller Objekte in einem Level zu jeder Zeit gespeichert werden. Auf die jeweils verwendete Abstrahierung gehen wir später noch einmal ein. Als erstes werden wir nun einige Grundlagen der Komplexitätstheorie betrachten, welche notwendig sind, um dem Thema folgen zu können, und vorangegangene Arbeiten auf dem Gebiet der Komplexität von Computerspielen betrachten. Dann werden wir die Ergebnisse von Giovanni Viglietta [Vig14] an dem bisher unbetrachteten Spiel „Super Mario World 2: Yoshi's Island“ anwenden und zeigen, dass besagtes Spiel NP-schwer ist. Im zweiten Teil der Arbeit werden wir zwei von mir entwickelte Theoreme betrachten, welche angelehnt an den Metatheoremen von Viglietta nicht die Komplexität eines konkreten Spiels betrachten, sondern verschiedene Spielelemente charakterisieren, deren Auftreten ein Spiel NP-schwer machen. Dies ermöglicht es uns, die Komplexität einer ganzen Klasse von Spielen auf einmal zu betrachten. Zuletzt werden wir diese Ergebnisse noch an einem weiteren Spiel anwenden. Bei dem Spiel handelt es sich um „Super Mario Bros.“, welches zwar bereits von Greg Aloupis et al. [ADGV15] untersucht worden ist, bei dessen Komplexitätsbestimmung jedoch ein essenzielles Spielelement außer Acht gelassen wurde. Bei diesem Spielelement handelt es sich um einen Timer, bzw. eine Stoppuhr. Der Spieler muss das Level abschließen, bevor der Timer abgelaufen ist. In dem Beweis von Greg Aloupis [ADGV15] wird dieser Timer nicht erwähnt, es ist aber zumindest eine Anpassung des Timers nötig, um beliebig große Level konstruieren zu können, die noch lösbar sind. In dem von uns vorgestellten Beweis ist dieser Timer maßgeblich an der NP-Schwere des Spiels beteiligt. Zuletzt werden die Ergebnisse diskutiert und es werden Anreize für weitere Arbeiten in diesem Bereich gegeben.
Sie sehen also, die Komplexitätstheorie kann alles andere als langweilig und trocken sein.

Kapitel 2

Grundlagen

In diesem Kapitel möchten wir zunächst einige Grundlagen der Komplexitätstheorie betrachten, die zum Verständnis der nachfolgenden Kapitel erforderlich sind. Die Komplexitätstheorie befasst sich damit Probleme anhand ihres zur Lösung benötigten Ressourcenbedarfs in Komplexitätsklassen einzuteilen. Dabei werden vor allem zwei Ressourcen betrachtet, die Laufzeit und der Platzbedarf einer Maschine, die das Problem entscheidet. Sowohl die Laufzeit, als auch der Platzbedarf einer Maschine wird in der Länge der Eingabe gemessen. Probleme werden hierbei formal als Sprachen aufgefasst.
Das in der Komplexitätstheorie betrachtete Berechnungsmodell ist die Turingmaschine. Eine Turingmaschine besteht aus einem Schreib- und Lesekopf, einem endlichen Zustandsgraphen und einem unendlichen Band, auf dem sie in einem Schritt genau ein Feld bearbeiten kann. Zur genauen Definition siehe „Computers and Intractability: A Guide to the Theory of NP-completeness“ von Michael Garey und David Johnson [GJ79]. Die Laufzeit einer Turingmaschine M auf einer Eingabe w ist die Anzahl der Schritte, die M auf w ausführt, bevor M hält. Der Platzbedarf von M ist die Anzahl der Felder auf dem Band, die M angesetzt auf w während der Berechnung besucht. Bei der Berechnung können Turingmaschinen entweder deterministisch oder nichtdeterministisch vorgehen. Bei deterministischen Turingmaschinen ist der nächste Schritt zu jedem Zeitpunkt eindeutig bestimmt. Das heißt jede Konfiguration der Turingmaschine während der Berechnung hat genau eine mögliche Nachfolgekonfiguration. Nichtdeterministische Turingmaschinen können hingegen gleichzeitig mehrere mögliche Nachfolgekonfigurationen haben. Diese Überlegungen führen uns zu zwei wichtigen Komplexitätsklassen, den Klassen P und NP. Die Klasse P umfasst alle Probleme, die von einer deterministischen Turingmaschine in polynomieller Laufzeit gelöst werden können. NP ist die Menge aller Probleme, die von einer nichtdeterministischen Turingmaschine in polynomieller Laufzeit gelöst werden können. Über die Beziehung von P und NP weiß man bisher nur, dass $\mathrm{P} \subseteq \mathrm{NP}$ gilt. Die Frage, ob $\mathrm{P} = \mathrm{NP}$ ist, gehört zu den bedeutendsten offenen Fragen der Informatik

und ist bis heute ungeklärt. Es wird jedoch vermutet, dass keine Gleichheit besteht.
Neben den Klassen P und NP soll an dieser Stelle noch die Klasse PSPACE erwähnt werden, welche alle Sprachen, die von Turingmaschinen mit polynomiellem Platzbedarf erkannt werden, umfasst.

Die Einteilung von Problemen in Komplexitätsklassen weckt den Wunsch, Probleme mathematisch miteinander vergleichen zu können. Für diese Aufgabe steht uns das Hilfsmittel der Reduktion zur Verfügung. Die Idee der Reduktion ist es, die Fragestellung für ein Problem A in eine Fragestellung für ein Problem B umzuwandeln und somit das Problem A durch das Problem B zu entscheiden. Ist A ein komplexitätstheoretisch schweres Problem, so ist B ebenfalls ein schweres Problem, da wir ja in der Lage sind, A durch B zu entscheiden.
Formal ist eine Reduktion eine Funktion und wie folgt definiert:

Definition 1 ([GJ79])**.**

Seien Σ und Γ endliche Alphabete und $A \subseteq \Sigma^*$ und $B \subseteq \Gamma^*$ zwei Sprachen.[1] *Eine totale und berechenbare Funktion $f : \Sigma^* \to \Gamma^*, w \mapsto f(w)$ heißt Reduktion von A nach B, falls gilt:*

$$\forall w \in \Sigma^* : w \in A \Leftrightarrow f(w) \in B$$

Ist f in polynomieller Zeit berechenbar, so heißt f Polynomzeitreduktion.
Wir schreiben

$$A \leq^m_{pol} B.$$

Wir werden uns vor allem mit der Komplexitätsklasse NP befassen. Innerhalb einer Komplexitätsklasse unterscheiden wir nochmals Probleme, von denen wir wissen, dass sie zu den schweren Problemen in dieser Klasse gehören, von den anderen Problemen der Klasse. Eine Sprache L heißt NP-*schwer* oder NP-*hart*, falls für jede Sprache in NP eine Polynomzeitreduktion auf L existiert. Ist L zusätzlich noch in NP enthalten, so nennen wir L NP-*vollständig*.
Lässt sich eine NP-schwere Sprache A mit einer Polynomzeitreduktion auf eine Sprache B reduzieren, so ist B ebenfalls NP-schwer, da die Hintereinanderausführung zweier Polynomzeitreduktionen ebenfalls eine Polynomzeitreduktion ist.
Unter der Annahme, dass P $\neq$ NP gilt, existieren somit keine Algorithmen mit polynomieller Laufzeit für NP-schwere Probleme.

[1] Die Sprache Σ^* ist die Menge alle Wörter über dem Alphabet Σ. Die Sprache Γ^* ist die Menge alle Wörter über dem Alphabet Γ.

Ein NP-schweres Problem, das bei einer unär kodierten Eingabe in polynomieller Zeit deterministisch gelöst werden kann, heißt *schwach*-NP-schwer [Sch10].

Neben den vorgestellten Grundlagen wird ein grundlegendes Verständnis der theoretischen Informatik vorausgesetzt, wie es in der Grundvorlesung „Theoretische Informatik“ vermittelt wird. Als Referenz empfehlen wir das Buch „Introduction to Automata Theory, Languages and Computation“ von John Hopcroft [Hop79]. Zudem werden wir graphentheoretische Grundlagen verwenden, welche beispielsweise in dem Buch „Graphentheorie“ von Reinhard Diestel [Die96] nachgelesen werden können.

Kapitel 3

Themenverwandte Arbeiten

Ihren Anfang fand die Komplexitätsbestimmung von Spielen in den achtziger Jahren bei der Analyse klassischer zwei-Personen Brettspiele, wie *Go* 1983 [Rob83] und *Schach* 1984 [Rob84] durch John M. Robson. Obwohl Nintendo bereits 1983 mit dem Nintendo Entertainment System die erste erfolgreiche Spielkonsole[1] auf den europäischen und amerikanischen Markt brachte, sollte es noch einige Jahre dauern, bis elektronische Computerspiele ihre Beachtung in der Komplexitätstheorie fanden.
Im Jahre 1999 wurde von Joseph Culberson das Computerspiel *Sokoban* untersucht. Er stellte fest, dass das Spiel PSPACE-vollständig ist, indem er mit den Möglichkeiten des Spiels eine linear platzbeschränkte Turingmaschine simulierte.
Ein Jahr später widmete sich Erik D. Demaine dem Gebiet der Komplexität von Computerspielen und trug mit zahlreichen Artikeln maßgeblich zum Fortschritt der Forschung auf diesem Gebiet bei. In seiner 2000 veröffentlichten Arbeit zeigte er zusammen mit Martin L. Demaine und Joseph O'Rourke, dass die Spiele *PushPush* und *Push-1* NP-schwer sind [DDO00]. Vier Jahre später fand er zusammen mit Michael Hoffmann und Markus Holzer heraus, dass die verallgemeinerte Form *PushPush-k* PSPACE-vollständig ist [DHH04].
Im Jahre 2000 wurde auch das bekannte Spiel *Minesweeper* untersucht und dessen NP-Vollständigkeit gezeigt [Kay00]. Drei Jahre später inspirierte dieses Ergebnis Lourdes Pena Castillo und Stefan Wrobel dazu, ein „General Purpose Learning System" zu trainieren, um eine Strategie zu entwickeln, Minesweeper zu lösen.
In dem 2001 erschienen Paper „Playing Games with Algorithms: Algorithmic Combinatorial Game Theory" geben Erik D. Demaine et al. eine umfassende Übersicht über die Ergebnisse, die auf dem Gebiet der Brettspiele und Computerspiele bis dahin gefunden wurden. Darunter sind auch Spiele wie *Hex*, *Cops and Robbers*, *Dame*, *Peg Solitaire* und *Schiebepuzzle* wie das

[1] Bis heute fast 62 Millionen verkaufte Exemplare[NES16].

15 puzzle [Dem01].
Zwei Jahre später widmeten sich Erik D. Demaine, Susan Hohenberger und David Liben-Nowell dem Computerspiel *Tetris* und zeigten, dass dieses nicht nur NP-vollständig, sondern auch schwer zu approximieren ist [DHLN03].
Im Jahr 2004 wurde zum ersten Mal das Computerspiel *Lemmings* untersucht. Während 2004 Graham Cormode noch behauptete, dass Lemmings NP-vollständig sei [Cor04], wurde 2010 von Michal Forišek gezeigt, dass es Instanzen in Lemmings gibt, die eine exponentiell große Lösung verlangen [For10]. Hierbei blieb jedoch offen, ob Lemmings sogar PSPACE-vollständig ist. Diese Frage wurde dann 2015 von Giovanni Viglietta beantwortet, der zeigte, dass Lemmings PSPACE-vollständig und als Optimierungsproblem APX-schwer ist [Vig15].
Eine weitere Übersicht über Ergebnisse in diesem Bereich, welche sich mehr auf sogenannte *Puzzle Games* oder zu Deutsch *Knobelspiele* konzentrierte, wurde 2008 von Graham Kendall, Andrew Parkes und Kristian Spoerer gegeben [KPS08]. Hierunter befanden sich Spiele wie *Blocks World*, *Clickomania*, *KPlumber*, *Nurikabe*, *Rush Hour*, *Solitaire* und *Sudoku*.
Während bisher einzelne, konkrete Spiele untersucht wurden, machten Erik D. Demaine und Robert A. Hearn erste Schritte, um allgemeinere Strukturen zu identifizieren, die es erleichtern, konkrete Spiele mit ihnen zu untersuchen. So entwickelten sie in ihrer 2008 veröffentlichen Arbeit [DH08] ein auf einem gerichteten Graphen basierendes Framework, bei dem der Spieler die Orientierung von Kanten ändern kann, solange gewisse Flussbedingungen in den Knoten erfüllt sind. Abhängig von der Anzahl an Spielern und der zugelassenen Größe des Graphen liefert dieses Framework Ergebnisse für verschiedene Komplexitätsklassen, von P bis NEXPTIME. Schließlich wendeten sie ihr Framework unter anderem auf die Spiele *Sokoban*, *Rush Hour*, *Pushing Blocks*, *Amazons* und *Konane* an.
Auch Michael Forišek verfolgte in seiner 2010 veröffentlichten Arbeit [For10] den Ansatz sogenannte Metatheoreme zu entwickeln und damit gleich mehrere Spiele auf einmal zu klassifizieren. Bei den Metatheoremen wurden verschiedene Spielelemente charakterisiert, deren Auftreten in einem Spiel diesem eine gewisse Komplexität verleihen, da sie die Konstruktion von „schweren“ Instanzen ermöglichen. Michael Forišeks Metatheoreme fanden abschließend bei den Spielen *Prince of Persia* und *Commander Keen* Anwendung.
Die Idee der Metatheoreme wurde drei Jahre später von Giovanni Viglietta aufgegriffen und ausgearbeitet [Vig14]. Er präsentierte mehrere Metatheoreme, die sich besonders für die Analyse von Jump&Run-Spielen, aber auch von verschiedenen Puzzle-Spielen, eignen. Die damit untersuchten Computerspiele umfassen unter anderem *Lode Runner*, *Lemmings*, *Pac-Man*, *Pipe Mania*, *Prince of Persia*, *Starcraft* und *Tron*. Die Arbeit von Giovanni Viglietta war maßgeblich Inspiration für die vorliegende Arbeit und soll im nächsten Kapitel noch genauer vorgestellt werden.

Im Jahre 2015 widmeten sich Greg Aloupis, Erik D. Demaine, Alan Guo und Giovanni Viglietta den berühmtesten klassischen Nintendo Spielen [ADGV15], darunter *Super Mario Bros.*, *Donkey Kong Country*, *The Legend of Zelda*, *Metroid* und *Pokémon*.
Das Spiel *Super Mario Bros.* wurde von Erik D. Demaine, Giovanni Viglietta und Aaron Williams 2016 noch einmal differenzierter untersucht [DVW16]. Hierbei wurde unter anderem die Kodierung des Spiels und der sichtbare Fensterbereich betrachtet, sowie unterschieden, ob die Aufgabe des Spielers darin besteht, ein einzelnes Level oder eine Sequenz von Leveln zu lösen.
Durch die komplexitätstheoretische Untersuchung von Computer- und Brettspielen wurden diese Spiele auch in anderen Bereichen interessant. Zum Beispiel wurden verschiedene Lernsysteme auf Brettspiele wie *Go* [RFB00], [BDR98], [SDS94], *Schach* [Sam59], [Mor96], *Shogi*, [NISI98] und *Backgammon* [Tes95], [Tes89] angesetzt. Aber auch Computerspiele wie *Minesweeper* [CW03], *Tetris* [SL06] und *Super Mario Bros.* [TSKY13] wurden trainiert.

Kapitel 4

„Gaming Is a Hard Job, but Someone Has to Do It!“

Hauptinspiration für diese Arbeit ist die Arbeit von Giovanni Viglietta auf dem Gebiet der Komplexität von Computerspielen. In seinem 2014 erschienen Artikel „Gaming Is a Hard Job, but Someone Has to Do It!“ [Vig14] weicht er von dem bisher üblichen Vorgehen, die Komplexität eines speziellen Spiels zu untersuchen, ab und identifiziert stattdessen Spielelemente und Mechaniken, deren Auftreten einem Spiel eine komplexitätstheoretische Härte verleihen. Diese Ergebnisse präsentiert er in mehreren Metatheoremen, die sich jeweils auf verschiedene Kombinationen von Spielelementen beziehen. Dabei konzentriert sich Giovanni Viglietta auf das Genre der Jump&Run- und Puzzle-Spiele. Zu erstgenanntem Genre gehören hauptsächlich Geschicklichkeitsspiele, bei denen der Spieler die Aufgabe hat, einen Avatar durch eine zweidimensionale Spielwelt zu einem dedizierten Ausgangspunkt zu lenken und dabei Hindernissen auszuweichen.
Da sich die Spielaufgabe bei Spielen dieses Genres meist darauf beschränkt, einen optimalen Weg durch eine Spielinstanz (im Folgenden Level genannt) zu finden, konnte Giovanni Viglietta die meisten seiner Theoreme durch Reduktionen des NP-vollständigen Graphen-Problems Hamiltonian Circuit beweisen.
Das Besondere an seiner Arbeit ist, dass nicht mehr ein konkretes Spiel untersucht wird, sondern die essenzielle Mechanik hinter dem Spiel, welche sich wiederum in anderen Spielen wiederfinden lässt. Somit ist es Giovanni Viglietta gelungen, die Komplexität vieler Spiele auf einmal zu ermitteln. Um die Komplexität eines konkreten Spiels zu bestimmen, bleibt nun lediglich zu zeigen, dass sich die in Vigliettas Metatheoremen beschriebenen Elemente und Mechaniken mit den Bausteinen, die dieses konkrete Spiel bietet, realisieren lassen und es somit möglich ist, mit den Bausteinen des Spiels „schwere“ Instanzen zu konstruieren. Da bislang die Komplexitätsbestimmung eines konkreten Spiels eine speziell für dieses Spiel geeignete Reduktion verlangte

und sich diese Reduktion von Spiel zu Spiel stark unterscheiden kann, bieten Vigliettas Metatheoreme einen uniformeren Ansatz zur Komplexitätsbestimmung.
Wir wollen nun im Folgenden eines von Vigliettas Metatheoremen vorstellen und dieses anschließend bei der Bestimmung der Komplexität des Spiels „Super Mario World 2: Yoshi's Island“ verwenden, zu dem uns noch keine Ergebnisse bekannt sind. Wir beschränken uns hierbei auf ein Metatheorem. Der interessierte Leser sei für weitere Theoreme auf den Artikel von Giovanni Viglietta [Vig14] verwiesen. Danach werden wir zwei von mir entwickelte Metatheoreme betrachten, welche ebenfalls Spiele aus dem Jump&Run-Genre behandeln.

4.1 Vigliettas Metatheorem

Wie zuvor beschrieben, behandeln Vigliettas Metatheoreme Kombinationen von Spielelementen. Das nachfolgende Theorem verbindet die Elemente *Türen, Einbahnstraßen, einsammelbare Schlüssel* und *zwingende Lokationen*.
Türen können geschlossen oder offen sein, sie können nur passiert werden, wenn sie geöffnet sind. Im geschlossenen Zustand dienen sie dazu, einen Weg zu versperren. Türen werden durch *Schlüssel* geöffnet. Eine geöffnete Tür bleibt für die Dauer des Levels geöffnet. Ein Schlüssel wird beim Öffnen einer Tür verbraucht. *Einsammelbare* Schlüssel können von dem Avatar an einer Stelle aufgenommen werden und zu einer anderen Stelle, an der sich zum Beispiel eine Tür befindet, mitgenommen werden. Dabei kann der Avatar nur einen Schlüssel gleichzeitig tragen.
Einbahnstraßen sind Wegabschnitte, die der Avatar nur in eine Richtung passieren kann.
Zwingende Lokationen sind besondere Positionen in einem Level, die der Avatar besuchen muss, um das Level zu gewinnen. Diese können zum Beispiel durch Gegenstände, die in einem Level eingesammelt werden müssen, implementiert sein.
Um ein Spiel überhaupt komplexitätstheoretisch untersuchen zu können, müssen wir eine abstrahierte Version des Spiels betrachten, um die Einschränkung der Realisierung des Spiels auf einem endlichen Rechner zu umgehen. Viglietta abstrahiert hierbei die Größe eines Levels, indem er Spiele mit beliebig großen Leveln betrachtet. Zudem nimmt er an, dass die Zustände aller Objekte in einem Level zu jeder Zeit gespeichert werden und das gesamte Level für den Spieler gleichzeitig sichtbar ist. Fasst man Computerspiele als Entscheidungsprobleme auf, so betrachtet Giovanni Viglietta das folgende Entscheidungsproblem.

Definition 2 (Entscheidungsproblem-Viglietta)**.**

Eingabe:
Ein Level mit zwingenden Lokationen, Türen, Schlüsseln und Einbahnstraßen, einer Startposition und, falls erforderlich, einer Zielposition.
Frage:
Ist es möglich von der Startposition aus alle zwingenden Lokationen zu erreichen und, falls erforderlich, zur Zielposition zu gelangen?

Das betrachtete Metatheorem lautet damit:

[Vig14, Metatheorem 3]

Ein Spiel, das Türen, Einbahnstraßen, einsammelbare Schlüssel und zwingende Lokationen enthält, ist NP-schwer.

Beweis:

Der Beweis wird nach Giovanni Viglietta [Vig14] durch eine Reduktion von einer speziellen Version des NP-vollständigen Hamiltonian Circuit Problems [GJ79] geführt. Dabei handelt es sich um das immer noch NP-schwere Hamiltonian Circuit Problem für gerichtete planare Graphen, deren Knoten entweder genau eine eingehende und zwei ausgehende oder genau zwei eingehende und eine ausgehende Kante besitzen [P+79]. Das Hamiltonian Circuit Problem ist wie folgt definiert [GJ79]:

Definition 3 (Hamiltonian Circuit Problem)**.**

Eingabe:
Ein Graph $G = (V, E)$.
Frage:
Enthält G einen Hamiltonian Circuit? Ein Hamiltonian Circuit ist eine Ordnung $\langle v_1, v_2, \ldots, v_n \rangle$ der Knoten von G, mit $n = |V|$, $\{v_n, v_1\} \in E$ und für alle i mit $1 \leq i < n$ gilt $\{v_i, v_{i+1}\} \in E$.

Kommen wir nun zur Reduktion. Sei $G = (V, E)$ ein gerichteter Graph mit obigen Eigenschaften. Wir wollen nun aus G eine Spielinstanz bauen, die genau dann gewinnbar[1] ist, wenn G einen Hamiltonian Circuit enthält. Die Reduktion verläuft wie folgt. Wir wählen einen beliebigen Knoten $v \in V$, der zwei eingehende und eine ausgehende Kante besitzt. An diesen Knoten hängen wir über eine neue ausgehende Kante einen neuen Knoten u an und erzeugen anschließend eine planare Darstellung des Graphen mit Spielbausteinen. Hierbei wird in jeden Knoten eine zwingende Lokation platziert und jede gerichtete

[1] Eine gewinnbare Spielinstanz ist ein Level, in dem es dem Spieler möglich ist, alle Bedingungen zu erfüllen, um das Level zu gewinnen.

Kante wird durch eine Einbahnstraße realisiert. Der Knoten v wird zur Startposition des Avatars und falls von dem Spiel eine Zielposition gefordert wird, so wird diese in u platziert. Der Knoten u muss als letztes erreicht werden, da u keine ausgehenden Kanten besitzt. Nun wird noch in jede Einbahnstraße, außer der zwischen v und u, eine geschlossene Tür und in jeden Knoten, außer u, ein Schlüssel platziert.
Nachdem der erste Schlüssel in v aufgenommen wurde, wird jedes Mal, wenn der Avatar eine neue Position erreicht, ein Schlüssel verbraucht (außer bei u). Dies bedeutet insbesondere, dass, wenn der Avatar eine bereits besuchte Position (außer v) über einen durch die Einbahnstraßen gezwungenermaßen anderen Pfad noch einmal besucht, hierbei ein Schlüssel verbraucht, aber kein neuer Schlüssel vorgefunden wird. Der Avatar ist dann nicht mehr in der Lage, Türen zu öffnen, und da sich weitere neue Schlüssel nur noch hinter verschlossenen Türen befinden, ist es dem Avatar nicht mehr möglich, die bisher noch nicht besuchten Lokationen (und insbesondere die Zielposition) zu besuchen. Somit ist es dem Spieler genau dann möglich, das Level erfolgreich abzuschließen, wenn er das Level so durchqueren kann, dass er auf dem Weg von v nach u jede Lokation genau einmal besucht. Dies ist nach Konstruktion genau dann der Fall, wenn G einen Hamiltonian Circuit enthält. □

4.2 Super Mario World 2: Yoshi's Island

Das Spiel „Super Mario World 2: Yoshi's Island“ ist ein 1995 von Nintendo veröffentlichtes Jump&Run Spiel für das Super Nintendo Entertainment System [Yos16b]. Der Spieler steuert die dinosaurierartige Figur *Yoshi*, welche in Abbildung 4.1 zu sehen ist, durch 48 Level [Yos16c]. Yoshi eskortiert dabei *Baby Mario* zu dessen Bruder *Luigi*, die in Abbildung 4.2 abgebildet sind. Die zu durchquerenden Level sind voller Gefahren und Gegner, die Yoshi überwinden muss. Ihm steht hierzu die Fähigkeit, auf kurze Distanz bestimmte Sorten von Gegnern mit seiner Zunge zu fangen und zu verschlucken, zur Verfügung. Ein verschluckter Gegner kann dann per Tastendruck in ein Ei umgewandelt werden. Ein Gegner wird hierbei zu genau einem Ei. Yoshi kann bis zu sechs Eier gleichzeitig hinter sich herziehen und noch einen Gegner im Mund behalten, um ihn später zu einem Ei zu machen. Die Eier dienen Yoshi als Munition. Sie können auf Gegner oder spezielle Gegenstände abgeschossen werden. Der Spieler kann hierbei den Abschusswinkel durch einen getimten Tastendruck beeinflussen. Mit den Eiern kann Yoshi Gegner, wie die *Piranha Plant* aus Abbildung 4.4 [Pir16], besiegen, gegen die er sonst nichts ausrichten könnte, oder spezielle Gegenstände einsammeln bzw. aktivieren, an die er sonst nicht herankäme.

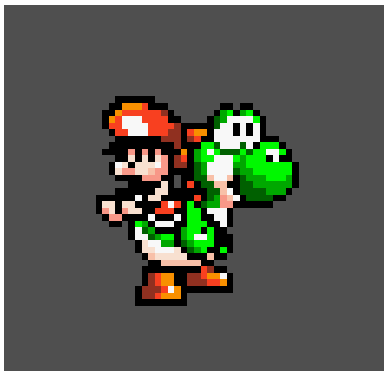

Abbildung 4.1: Die Hauptfigur Yoshi mit Baby Mario auf dem Rücken [Yos16a].

Nachdem ein Level erfolgreich abgeschlossen wurde, übernimmt ein anderer Yoshi, in einer anderen Farbe, die Eskorte von Baby Mario.

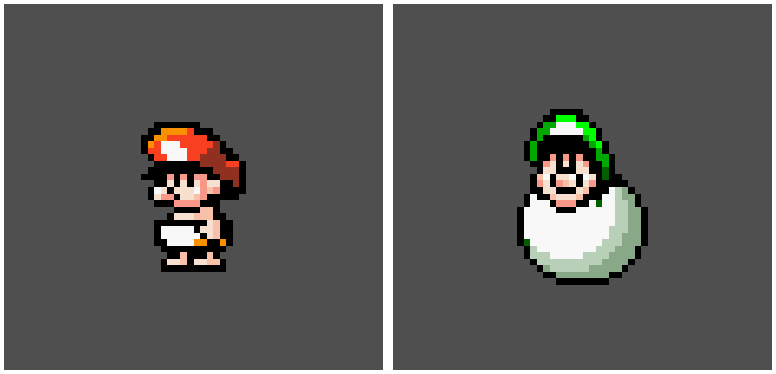

Abbildung 4.2: Links ist Baby Mario zu sehen und rechts sein entführter Bruder Baby Luigi [Yos16a].

Die häufigste Gegnersorte in „Super Mario World 2: Yoshi's Island" ist der so genannte *Shy Guy* [Shy16], wie er in Abbildung 4.3 zu sehen ist. Bei dem Shy Guy handelt es sich um einen verspeisbaren Gegner, wohingegen die *Piranha Plant* [Pir16] in Abbildung 4.4 von Yoshi nicht verschluckt werden kann. Sie kann nur durch einen Treffer mit einem Ei besiegt werden.

Abbildung 4.3: Der Shy Guy Gegner tritt in verschiedenen Farben auf und kann von Yoshi gefressen werden [Yos16a].

Abbildung 4.4: Die Piranha Plant kann nur durch Beschuss mit einem Ei besiegt werden und ist nicht verspeisbar [Yos16a].

Eine andere Eigenschaft von Yoshi ist, dass er sich an bestimmten Stellen in andere Formen verwandeln kann. Eine dieser Formen ist Hubschrauber-Yoshi. Um sich verwandeln zu können, muss Yoshi eine große Seifenblase, in der sich eine Miniaturform der Gestalt, in die er sich verwandeln will, befindet, berühren. Eine solche Seifenblase für die Hubschrauber-Gestalt ist in Abbildung 4.5 zu sehen. Hat Yoshi sich verwandelt, so bleibt ihm eine gewisse Zeit, um in der neuen Form das Symbol zu erreichen, welches ihn zurück verwandelt. Schafft er dies nicht in der ihm zur Verfügung stehenden Zeit, so wird er zu der Position, an der er sich verwandelt hat, zurückgezogen. Die neuen Formen verleihen Yoshi vorübergehend gewisse Fähigkeiten. So kann Yoshi in der Hubschrauberform fliegen und dadurch große Abgründe überwinden. Yoshi kann sich nur bei dem dafür vorgesehenen Symbol wieder zurückverwandeln und setzt seine Reise von dort aus fort. Das Rückwandel-Symbol für die Hubschrauberform ist in Abbildung 4.6 zu sehen.

Abbildung 4.5: Berührt Yoshi dieses Symbol, so nimmt er seine Hubschrauberform an [Yos16a].

Abbildung 4.6: In der Hubschrauberform muss Yoshi innerhalb kurzer Zeit dieses Symbol erreichen, um wieder seine normale Gestalt anzunehmen [Yos16a].

Der Spieler hat ein Level erfolgreich abgeschlossen, wenn er Yoshi zum Ausgang des Levels geführt hat.

4.3 Realisierung der Spielelemente

Wir werden nachfolgend nun die einzelnen Elemente aus obigem Metatheorem mit den durch das Spiel bereit gestellten Bausteinen realisieren und damit die NP-Schwere von „Super Mario World 2: Yoshi's Island“ zeigen.
Wir verallgemeinern das Spiel auf die übliche Weise dahingehend, dass wir beliebig große Level zulassen und davon ausgehen, dass das gesamte Level gleichzeitig zu sehen ist und somit der Zustand aller Objekte jederzeit gespeichert wird.
Die Realisierungen mancher Elemente werden durch Screenshots aus dem Spiel dargestellt. Diese stammen aus einer online emulierten Kopie des Spiels, die unter `http://www.playretrogames.com/2380-super-mario-world-2-yoshi-s-island` [Yos15] zu finden ist. Für andere Elemente mussten zuvor noch nicht implementierte Kombinationen von Bausteinen realisiert werden. Die hierzu verwendeten Sprites stammen von `https://www.spriters-resource.com/snes/yoshiisland/` [Yos16a].

Türen

Die Türen werden durch ein Gadget aus fleischfressenden (oder besser Yoshi-fressenden) Pflanzen, den Piranha Plants, realisiert. Abbildung 4.7 zeigt die Konstruktion. Der Korridor, in dem sich die Pflanze befindet, ist dabei so niedrig, dass Yoshi die Pflanze nicht überspringen kann. Er kann also nur passieren, wenn er die Pflanze besiegt hat. Da die fleischfressenden Pflanzen nur besiegt werden können, indem man sie mit einem Ei beschießt, kann unser Tür-Gadget nur geöffnet werden, wenn der Avatar im Besitz eines Eies ist. Da die Gegner, nachdem sie besiegt wurden, nicht wieder erscheinen, bleibt die Tür für den Rest des Levels geöffnet.

Abbildung 4.7: Tür mit Schlüssel [Yos15].

Einbahnstraßen

Einbahnstraßen lassen sich auf verschiedene Weisen realisieren. Durch Abgründe, die tiefer sind, als Yoshi nach oben springen kann, lassen sich Einbahnstraßen von oben nach unten realisieren. Durch einseitig durchspringbare Balken, wie sie in Abbildung 4.8 zu sehen sind, lassen sich Einbahnstraßen von unten nach oben erzielen. Der Avatar kann diese von unten durch Springen überqueren und kommt dann auf ihnen zum Stehen. Einmal überquert, entsprechen sie festem Boden und können nicht mehr von oben nach unten passiert werden. Des Weiteren existieren Bausteine, wie sie in Abbildung 4.9 zu sehen sind, die sich nur durch Annäherung von einer Seite öffnen lassen

und somit nur in eine Richtung passierbar sind. All diese Bausteine können zum Bau von Einbahnstraßen verwendet werden. Dabei kann man mit jedem Baustein eine Einbahnstraße für alle vier Richtungen des Levels bauen.

Abbildung 4.8: Einbahnstraße mit einseitig durchspringbarem Balken [Yos16a], [Yos15].

Abbildung 4.9: Einbahnstraße mit einseitig öffnendem Tor [Yos15].

Einsammelbare Schlüssel

Die einsammelbaren Schlüssel werden durch Instanzen der Gegner-Sorte, welche Yoshi verschlucken kann, realisiert. Abbildung 4.10 zeigt mit dem *Shy Guy* einen solchen Gegner. Diese Gegner können dann zu Eiern umgewandelt werden, mit welchem ein Tür-Gadget geöffnet werden kann. Da die Gegner in „Super Mario World 2: Yoshi's Island“, nachdem sie gefressen oder besiegt wurden, an den meisten Stellen nicht wieder erscheinen, steht für jede Tür nur genau

ein Schlüssel zur Verfügung. Alternativ zu den Gegnern als Schlüssel existieren in manchen Leveln besondere Blöcke, welche sich beim Dagegenspringen in genau ein Ei verwandeln. Dieses Ei kann dann ebenfalls aufgenommen und als Schlüssel verwendet werden. Bei der Benutzung des Blocks wird dieser verbraucht und erscheint nicht wieder. Ein solcher Block ist in Abbildung 4.11 zu sehen. Da diese Blöcke aber nur in bestimmten einzelnen Leveln auftreten, die verschluckbaren Gegner dagegen in jedem Level zu finden und elementar für das Spiel sind, wird die Realisierung der Schlüssel durch Gegner bevorzugt.[2]

Abbildung 4.10: Realisierung des Schlüssels [Yos15].

[2]Yoshi kann bis zu sechs Eier und einen, noch nicht in ein Ei verwandelten Gegner mit sich herum tragen, die einsammelbaren Schlüssel sind jedoch so gedacht, dass immer nur ein Schlüssel gleichzeitig getragen werden kann. Wir können die Tür- und Schlüssel-Gadgets daher auch so anpassen, dass Yoshi bei einem Schlüssel immer genau sieben Gegner vorfindet und verschluckt und bei einer Tür sieben Gegner hintereinander besiegen muss, also alle sieben verschluckten Gegner an einer Tür verwenden muss.

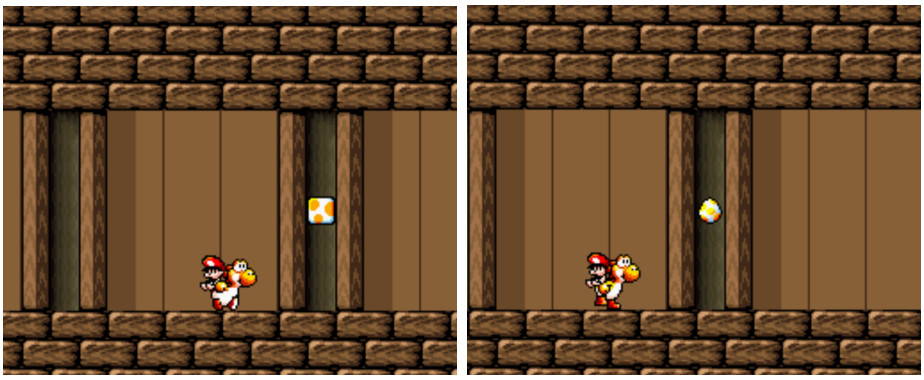

Abbildung 4.11: Alternative Schlüssel [Yos15].

Zwingende Lokationen

Die zwingenden Lokationen werden durch ein Gadget realisiert, welches direkt vor dem Ausgang platziert wird, und somit zwingend passiert werden muss, um das Level zu beenden. Abbildung 4.12 zeigt dieses Gadget. Dabei handelt es sich um einen langen, einseitig durchspringbaren horizontalen Balken, über dem in gewissen Abständen Piranha Plants von niedrigen Balken hängen. Die Pflanzen sind so tief, dass Yoshi sie nicht passieren kann, bevor er sie nicht mit einem Ei-Geschoss besiegt hat. Am Anfang des Korridors befindet sich eine Treppe, über die Yoshi von unten an den Anfang des Korridors gelangen kann. Dies ist der einzige Zugang zu dem Zielkorridor, was insbesondere bedeutet, dass Yoshi alle Pflanzen passieren muss, um zum Ziel zu gelangen. Nicht nur Yoshi, sondern auch die Eier können einseitig durchspringbare Balken von unten nach oben passieren. Dies bedeutet, dass Yoshi die Pflanzen von unten mit Eiern abschießen kann, bevor er sich selbst auf den Balken begibt. Die dazu benötigten Eier werden durch fressbare Gegner geliefert, die sich unterhalb des horizontalen Balkens in Kammern befinden, aus denen sie nicht entweichen können. Yoshi kann nun über treppenförmig angeordnete Balken in eine dieser Kammern gelangen, den dort gefangenen Gegner fressen, zu einem Ei verarbeiten und mit diesem Ei die Pflanze des Zielkorridors, welche sich oberhalb der Kammer befindet, abschießen. Die Pflanzen sind hierbei so weit auseinander platziert, dass es nicht möglich ist, mit einem Ei zwei Pflanzen zu treffen. Nachdem eine Pflanze des Zielkorridors besiegt wurde, kann Yoshi die Kammer, in dem sich der verspeiste Gegner befunden hat, über die Treppe wieder verlassen.
Die Kammern mit den Gegnern sind hier die zwingenden Lokationen, da Yoshi sie besuchen muss, um die fleischfressenden Pflanzen des Zielkorridors zu besiegen, gegen die er andernfalls nichts ausrichten kann. Die Pflanzen des Zielkorridors müssen zwingend besiegt werden, da Yoshi den Korridor andernfalls nicht passieren kann und er muss den Korridor passieren, da er sonst nicht zu dem Ausgang des Levels gelangen kann.

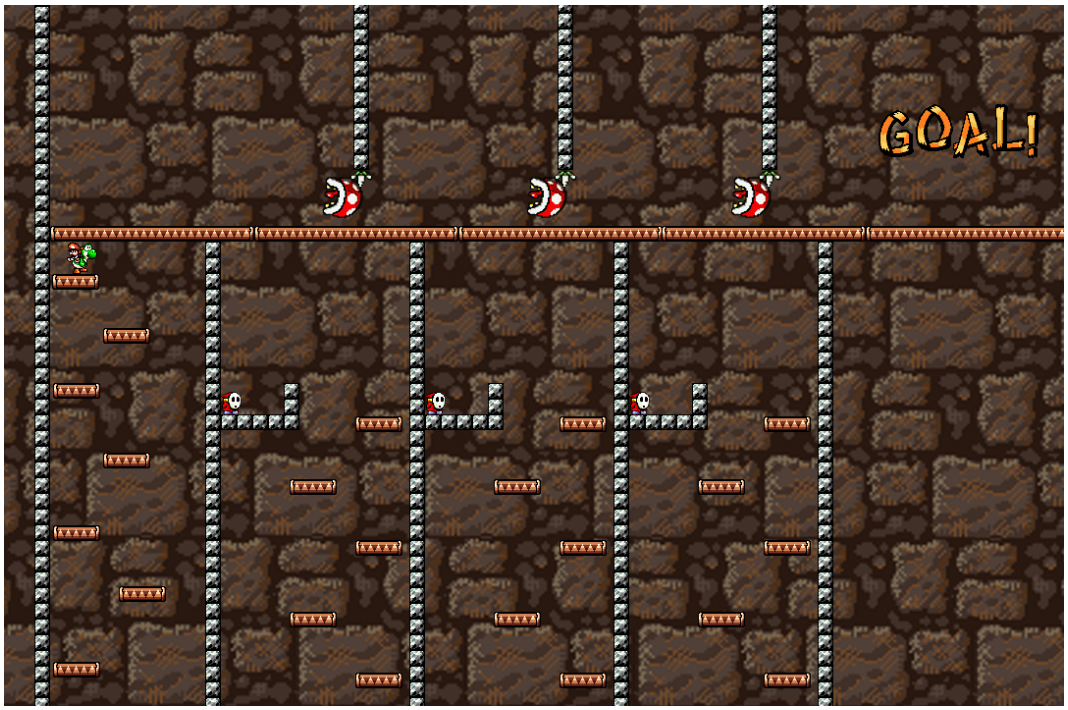

Abbildung 4.12: Realisierung der zwingenden Lokationen [Yos16a].

Kreuzungs-Gadget

Da wir die zwingenden Lokationen alle vor dem Ausgang des Levels platzieren, kann es zu Kreuzungen von Pfaden kommen, auch wenn der ursprüngliche Graph, von dem reduziert wird, planar ist. Um zu verhindern, dass der Avatar an diesen Kreuzungen die Richtung ändern kann, müssen wir zudem ein Kreuzungs-Gadget realisieren.
Bei der Realisierung des Kreuzungs-Gadgets nutzen wir Yoshis Fähigkeit, sich in andere Formen zu verwandeln. Im Speziellen benutzen wir die Hubschrauberform. Abbildung 4.13 zeigt das Kreuzungs-Gadget. Das Gadget kann entweder horizontal oder vertikal überquert werden. Für die vertikale Überquerung kann Yoshi die treppenförmig angeordneten Plattformen benutzen. Für die horizontale Überquerung muss sich Yoshi in seine Hubschrauberform verwandeln und kann dann die hohen Wände umfliegen, um sich auf der anderen Seite bei dem entsprechenden Symbol wieder in seine normale Form zu verwandeln. Die hohen Wände, welche den horizontalen Pfad unterbrechen, und der Korridor entlang der Treppe verhindern, dass Yoshi von dem horizontalen Pfad in den vertikalen gelangen kann und umgekehrt. Da Yoshi, wenn er seine Form gewandelt hat, innerhalb einer vorgegebenen Zeit das Rückwandel-Symbol erreichen muss, kann er auch in der Hubschrauberform nicht dem vertikalen Pfad folgen, da er nach kurzer Zeit wieder zurückgezogen werden würde.

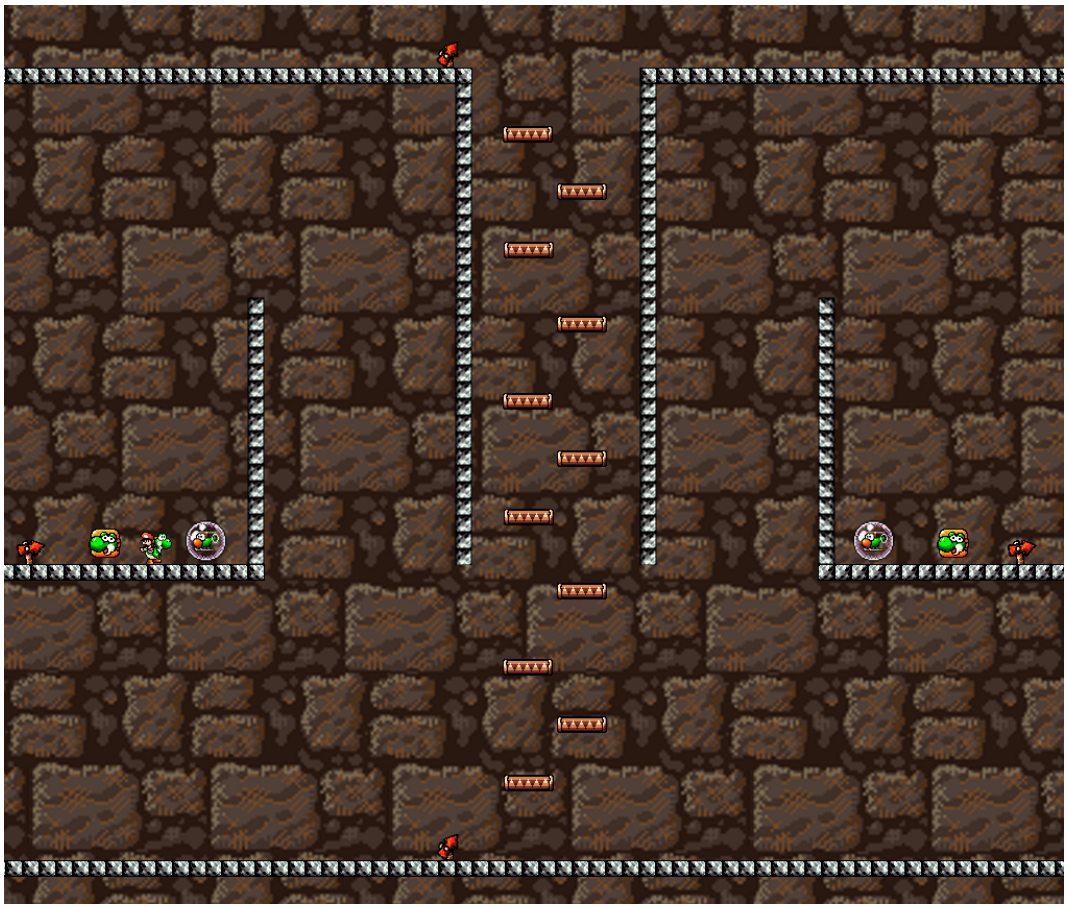

Abbildung 4.13: Ein Kreuzungs-Gadget, das entweder horizontal oder vertikal durchquert werden kann. Es kann jedoch keinen Richtungswechsel von horizontal zu vertikal und umgekehrt geben [Yos16a].

Da wir alle von dem Theorem geforderten Spielelemente realisieren konnten, können wir nun die Komplexität von „Super Mario World 2: Yoshi's Island" angeben.

Satz 1.

Die am Anfang des Kapitels beschriebene verallgemeinerte Form[3] *von „Super Mario World 2: Yoshi's Island" ist NP-schwer.*

Beweis:

Wir konnten alle geforderten Spielelemente des Metatheorems 3 aus Kapitel 4.1 mit den Bausteinen, die das Spiel „Super Mario World 2: Yoshi's Island" zur Verfügung stellt, realisieren. Somit ist das Spiel nach Vigliettas Metatheorem NP-schwer. □

[3]Beliebig große Level, gesamtes Level ist gleichzeitig zu sehen, Zustände aller Objekte werden jederzeit gespeichert.

Kapitel 5

Spielen unter Zeitdruck

Wir kommen nun zum Hauptteil der Arbeit. Angelehnt an den Metatheoremen von Giovanni Viglietta [Vig14] werden wir zwei Theoreme präsentieren, die sich mit dem Spielelement der Zeit befassen. Das erste Theorem behandelt die Elemente *zwingende Lokationen*, *Wurmlöcher* und eine *Stoppuhr*. Im Anschluss an das Theorem werden wir auf die Realisierung der Wurmlöcher und auf weitere technische Details eingehen. Das zweite Theorem beschäftigt sich mit Sammelobjekten und einer Stoppuhr. Im Rahmen dieses Theorems werden wir die Arbeit „Computational Complexity of Two-Dimensional Platform Games“ von Michal Forišek [For10] betrachten und ein Metatheorem aus dieser Arbeit, welches sich ebenfalls mit dem Spielelement der Zeit beschäftigt, diskutieren. Des Weiteren werden wir bezüglich dieses Theorems die Kodierung eines Spiels betrachten und auf den Artikel „Super Mario Bros. Is Harder/Easier than We Thought“ [DVW16], in welchem die von uns verwendete Kodierung eingeführt wird, eingehen. Als nächstes werden wir die Forderung der *uniformen Wege* diskutieren, die verlangt, dass sich der Avatar in jede Richtung gleich schnell bewegt. Zuletzt werden wir einige technische Details betrachten und die beiden Theoreme gegenüber stellen.

5.1 Theorem 1: Lokationen, Wurmlöcher und Stoppuhr

In der Literatur größtenteils unbehandelt blieb das Element der Zeit. In manchen Jump&Run-Spielen muss der Spieler ein Level innerhalb einer vorgegebenen Zeitspanne abschließen, um dieses zu gewinnen. In dem von mir entwickelten und im Folgenden vorgestellten Metatheorem wird dieses Spielelement verwendet, um weitere Kombinationen verschiedener Elemente anzugeben, deren Auftreten ein Spiel NP-schwer macht. Neben dem bisher

bekannten Element der *zwingenden Lokationen* und dem Element der Zeit, welches wir mit einer *Stoppuhr* bezeichnen wollen, werden wir noch ein weiteres Element betrachten. Bei diesem Element handelt es sich um *Wurmlöcher*, welche verschiedene Orte in einem Level verbinden. Dabei kann es sich entweder um eine gerichtete oder ungerichtete Verbindung handeln. Wir gehen davon aus, dass genau zwei Orte mit einem Wurmloch verbunden sind. Die Zeit, die ein Avatar benötigt, um ein Wurmloch zu passieren, ist immer gleich lang, egal wie weit die verbundenen Orte in dem Spiel voneinander entfernt sind. Wurmlöcher können auch Orte verbinden, die auf anderem Wege nicht verbunden sind. Wurmlöcher können auf verschiedene Weise realisiert werden. Zum einen durch *Röhren*, zu deren Benutzung man sich direkt neben oder auf der Röhre befinden und zusätzlich noch eine Eingabe machen muss, oder zum anderen durch *Tore*, die man beim Passieren automatisch benutzt.

Wie auch Giovanni Viglietta abstrahieren wir die Spiele dahingehend, dass wir beliebig große Level erlauben und das gesamte Level auf einmal zu sehen ist und somit die Zustände aller Objekte jederzeit gespeichert werden. Das betrachtete Entscheidungsproblem für ein festes Spiel lautet hierbei:

Definition 4 (Entscheidungsproblem 1)**.**

Eingabe:
Ein Level mit n zwingenden Lokationen, Wurmlöchern, einer Stoppuhr mit Zeit t, einer Startposition u und, falls erforderlich, einer Zielposition v.
Frage:
Ist es möglich, innerhalb der Zeit t, beginnend bei u, alle zwingenden Lokationen zu besuchen und, falls erforderlich, die Zielposition v zu erreichen?

Die Forderung, eine ausgezeichnete Zielposition als letztes zu erreichen, ändert hierbei nichts an der Komplexität des Problems. Der angegebene Beweis ist so konstruiert, dass eine Zielposition gefordert werden kann.

Kommen wir nun zu unserem ersten Hauptergebnis. Das nachfolgende Theorem ist, angelehnt an Giovanni Vigliettas Arbeit, als Metatheorem formuliert. Die in dem Theorem betrachtete Sprache ist das Entscheidungsproblem 1 aus Definition 4.

Theorem 1.

Ein Spiel, das zwingende Lokationen, Wurmlöcher und eine Stoppuhr enthält, ist NP-schwer.[1]

[1]Hierbei betrachten wir Spiele gemäß dem Entscheidungsproblem 1

Beweis:

Wir führen den Beweis durch eine Reduktion vom NP-vollständigen Hamiltonian Circuit Problem, welches selbst für 3-reguläre ungerichtete Graphen NP-vollständig ist [GJ79].

Wiederholung (Hamiltonian Circuit Problem)

Eingabe:
Ein Graph $G = (V, E)$.
Frage:
Enthält G *einen Hamiltonian Circuit? Ein Hamiltonian Circuit ist eine Ordnung* $\langle v_1, v_2, \ldots, v_n \rangle$ *der Knoten von* G*, mit* $n = |V|$*,* $\{v_n, v_1\} \in E$ *und für alle* i *mit* $1 \leq i < n$ *gilt* $\{v_i, v_{i+1}\} \in E$.

Reduktion

Die Reduktion ist wie folgt konstruiert. Zunächst überprüfen wir, ob es sich bei der Eingabe um einen korrekt kodierten Graphen handelt. Ist dies nicht der Fall, so bilden wir die Eingabe konstant auf eine unlösbare Spielinstanz, wie sie in Abbildung 5.1 zu sehen ist, ab. Der Kreis repräsentiert hierbei die Startposition des Avatars und der Stern eine zwingende Lokation. Offenbar ist es nicht möglich, von der Startposition aus den Stern zu erreichen, was insbesondere bedeutet, dass das Level nicht lösbar ist.

Abbildung 5.1: Unlösbare Spielinstanz. Die Zielposition rechts ist von der Startposition links aus nicht erreichbar.

Nehmen wir nun an, bei der Eingabe handelt es sich um einen korrekt kodierten Graphen $G = (V, E)$. Wir wollen nun aus G ein Level generieren, das genau dann innerhalb der durch die Stoppuhr vorgegebenen Zeit lösbar ist, wenn G einen Hamiltonian Circuit enthält. Dazu erzeugen wir für jeden Knoten $v \in V$ ein Gadget wie in Abbildung 5.2. Auf der linken Seite befinden sich die Wurmlöcher, welche die Knoten miteinander verbinden. Ein Wurmloch-Paar, bestehend aus einem Wurmloch des Knoten-Gadgets eines Knotens v_i und einem anderen Wurmloch des Knoten-Gadgets eines anderen Knotens v_j, realisiert dabei genau die Kante $\{v_i, v_j\}$ des Graphen. Nach der Benutzung

eines Wurmlochs betritt der Spieler das Level an der mit einem Kreis gekennzeichneten Stelle. Auf der rechten Seite des Gadgets wird eine zwingende Lokation platziert, die hier durch einen Stern dargestellt wird. Die Zeit, die der Avatar von den Wurmlöchern zur zwingenden Lokation benötigt, betrage l. O.B.d.A.[2] nehmen wir an, dass die für den Hin- und Rückweg benötigte Zeit gleich lang ist. Die Zeit, die der Avatar für die Durchquerung eines Wurmlochs benötigt, sei d. Nach Betreten des Gadgets benötigt der Avatar somit $2l + d$ Zeit, um den Stern einzusammeln, zurück zu den Wurmlöchern zu gelangen und ein neues Gadget über die Wurmlöcher zu erreichen.

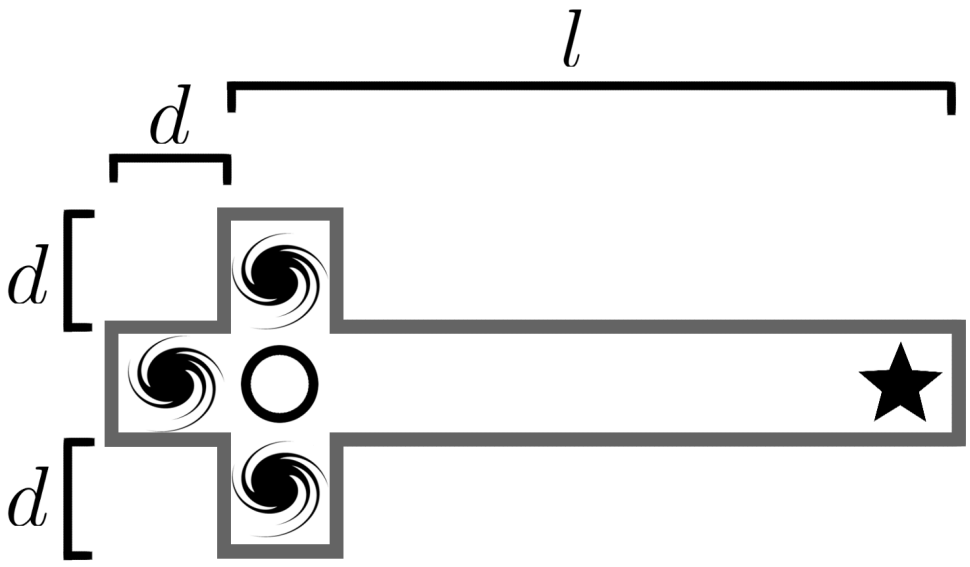

Abbildung 5.2: Gadget für einen beliebigen Knoten.

Um die Startposition des Avatars festzulegen, wählen wir einen beliebigen Knoten $p \in V$. An diesen Knoten hängen wir einen neuen Knoten $q \notin V$ über eine neue Kante an. p wird zur Startposition und falls eine Zielposition erforderlich ist, wird diese in q platziert. Das Gadget für p weicht etwas von den anderen Gadgets ab, da p im Gegensatz zu den anderen Knoten vier adjazente Kanten besitzt. Abbildung 5.3 zeigt das Gadget für p und Abbildung 5.4 zeigt das Gadget für q. Die drei Wurmlöcher auf der linken Seite des Startgadgets realisieren wie bei dem Gadget für beliebige Knoten die drei ursprünglichen Kanten von p, das Wurmloch auf der rechten Seite verbindet das Startgadget mit dem Zielgadget. Der Spieler startet an der mit einem Kreis markierten Position. Das schnellste Vorgehen ist, nach dem Start das Startgadget sofort durch eines der drei Wurmlöcher auf der linken Seite zu verlassen und den

[2]Siehe Abschnitt 5.2.4.

Stern erst am Ende, bei dem Besuch von q einzusammeln. Andernfalls würde der erste Abschnitt der Länge l drei- statt nur einmal besucht werden. Das Verlassen des Startgadget kostet d Zeit. Nachdem alle anderen Knoten besucht wurden, muss der Spieler noch einmal zu p zurückkommen, um q zu erreichen, da q von keinem anderen Knoten aus erreichbar ist.[3] Um das rechte Wurmloch zu erreichen und zu benutzen, benötigt der Spieler $2l + d$ Zeit, insgesamt also $2l+2d$. Das Zielgadget ist so konstruiert, dass dessen Durchquerung $2l$ benötigt. Die Position des Sterns des Zielgadgets ist auch gleichzeitig die Zielposition des Spiels, falls diese erforderlich ist. Um jeden der n Knoten, also jede zwingende Lokation, einmal zu besuchen, ist mindestens eine Zeit von $(n + 1) \cdot (2l + d)$ notwendig. Wir setzen also die Stoppuhr auf $(n + 1) \cdot (2l + d)$.

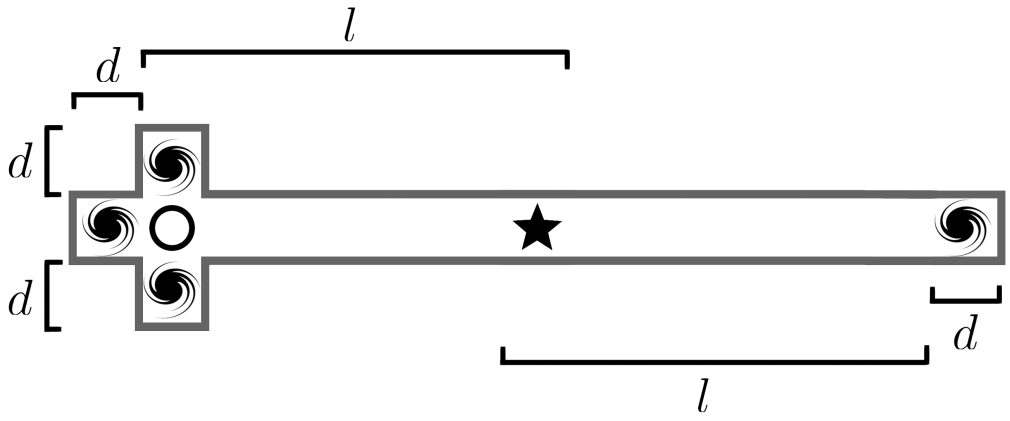

Abbildung 5.3: Gadget für die Startposition.

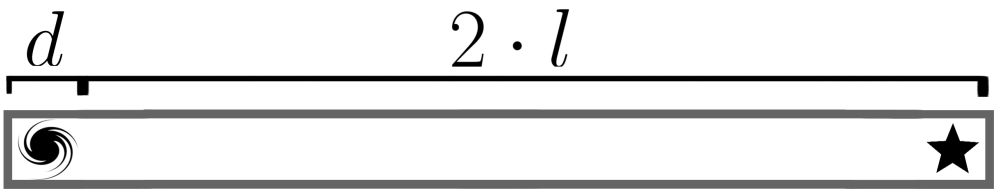

Abbildung 5.4: Gadget für die Zielposition.

Korrektheitsbeweis

Wir wollen nun zeigen, dass G genau dann einen Hamiltonian Circuit enthält, wenn das von uns konstruierte Level in der vorgegebenen Zeit lösbar ist.

[3]Falls das Spiel eine Zielposition fordert, so muss das Startgadget keine zwingende Lokation enthalten, da das Durchqueren des Startgadget bereits gefordert ist, da die Zielposition nur über das Startgadget erreichbar ist.

Ein Hamiltonian Circuit ist der kürzeste Pfad in einem Graphen, der vom Start-Knoten aus alle Knoten des Graphen genau einmal besucht und wieder zurück zum Start-Knoten führt. Für einen Graphen mit n Knoten hat ein Hamiltonian Circuit immer die Größe n. Das heißt, der kleinste Kreis, auf dem alle Knoten besucht werden, hat, falls er existiert, immer die Länge n.
Aufgrund der zwingenden Lokationen muss jedes Gadget mindestens einmal besucht werden, um das Level abzuschließen. Nach obiger Überlegung sind dazu n Kantenübergänge notwendig. Da wir in unserer Konstruktion noch einen zusätzlichen Ziel-Knoten an den Graphen angehängt haben, sind hier $n + 1$ Übergänge notwendig. Der Zielknoten muss als letztes besucht werden, da der einzusammelnde Stern $2l$ weit von dem Eingang des Wurmlochs entfernt ist. Würde der Spieler dieses Gadget nicht als letztes besuchen, so müsste er es wieder über das Wurmloch auf der linken Seite verlassen. In diesem Fall würde er jedoch $2 \cdot 2l$ Zeit auf dieses Gadget verbrauchen und nach obiger Überlegung für die Dauer der Besuche anderer Knoten, würde dies in eine Gesamtzeit $> (n+1) \cdot (2l+d)$ resultieren. Ebenso kann der Spieler kein zuvor besuchtes Gadget erneut besuchen, da dies mindestens zu einer Verzögerung von d führen würde. Es ist also nur möglich, das Level innerhalb von $(n + 1) \cdot (2l + d)$ abzuschließen, wenn für den Besuch aller zwingenden Lokationen nur $n + 1$ Kantenübergänge nötig sind. Dies ist genau dann der Fall, wenn der ursprüngliche Graph einen Hamiltonian Circuit enthielt.
Enthält der ursprüngliche Graph keinen Hamiltonian Circuit, so ist es entweder nicht möglich von jedem beliebigen Knoten aus alle anderen Knoten in einem Kreis zu erreichen, oder es sind hierzu mehr als n Kantenübergänge notwendig, was darin resultiert, dass mindestens ein Knoten mehrfach besucht wird. In beiden Fällen ist es nicht möglich, das Level innerhalb der durch die Stoppuhr vorgegebenen Zeit abzuschließen und dabei alle zwingenden Lokationen zu besuchen.

Laufzeit

Zu überprüfen, ob es sich bei der Eingabe um einen korrekt kodierten Graphen handelt, ist in linearer Zeit möglich. Für einen Graphen werden in unserer Reduktion $n + 1$ Gadgets konstanter Größe erzeugt. Die Berechnung oben beschriebener Reduktion ist somit durch eine linear in der Eingabe zeitlich beschränkte Turingmaschine möglich. Wir haben folglich eine totale und in polynomieller Zeit berechenbare Funktion gefunden, die eine Instanz des Hamiltonian Circuit Problems für 3-reguläre ungerichtete Graphen auf eine äquivalente Spiel-Instanz abbildet, unter Verwendung von zwingenden Lokationen, Wurmlöchern und einer Stoppuhr. Damit ist ein Spiel, welches diese Elemente enthält, mindestens NP-schwer. □

5.1.1 Anmerkungen

Wir möchten noch anmerken, dass das Zielgadget, falls eine Zielposition gefordert ist, nicht unbedingt die gleiche Länge wie die anderen Gadgets haben muss, da wir wissen, dass wir es in jedem Fall durchqueren müssen und folglich die Stoppuhr entsprechend anpassen können.
Wir können ebenfalls die Kreuzung mit den Wurmlöchern so modifizieren, dass sie nur von einer Richtung betreten werden. Dies kann zum Beispiel dann nötig sein, wenn das Spiel Schwerkraft realisiert und der Avatar ansonsten in das untere Wurmloch fallen würde. In diesem Fall verwenden wir Sprünge, um zwischen den Wurmlöchern zu wählen. Die Länge der Wegabschnitte l_1, l_2 und l_3 muss dann so gewählt werden, dass trotz der Sprünge die benötigte Zeit für l_1, l_2 und l_3 gleich lang ist. Abbildung 5.5 zeigt die beschriebene Konstruktion. Werden die Wurmlöcher durch Röhren, wie zum Beispiel in „Super Mario Bros.“ realisiert, so wird diese Modifikation nicht benötigt, da der Avatar auf den Röhren stehen kann, ohne sie zwangsläufig zu benutzen. Erst durch eine erneute Richtungsangabe in die Richtung der Röhre bestätigt der Spieler, dass er die Röhre benutzen will.

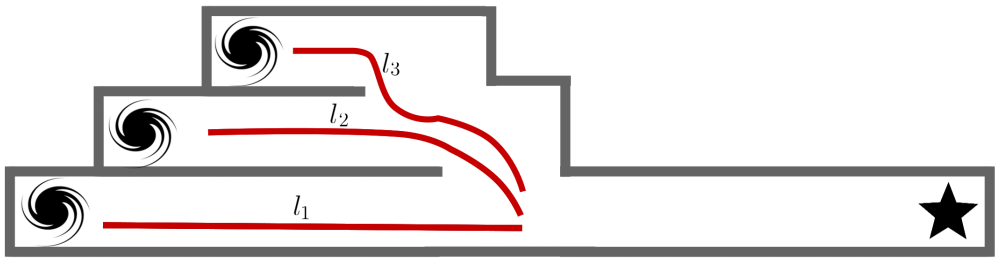

Abbildung 5.5: Modifiziertes Wurmloch-Gadget, bei dem die Wurmlöcher nur horizontal betreten werden.

Des Weiteren ist es irrelevant, ob die Wurmlöcher unidirektional oder bidirektional sind, also, ob sie nur in eine Richtung oder in beide Richtungen benutzt werden können. Sollten sie nur in eine Richtung benutzbar sein, so können wir das Level gemäß Abbildung 5.6 aufklappen und die Wurmlöcher auf der linken Seite als ankommende Wurmlöcher und die auf der rechten Seite als abgehende Wurmlöcher betrachten.

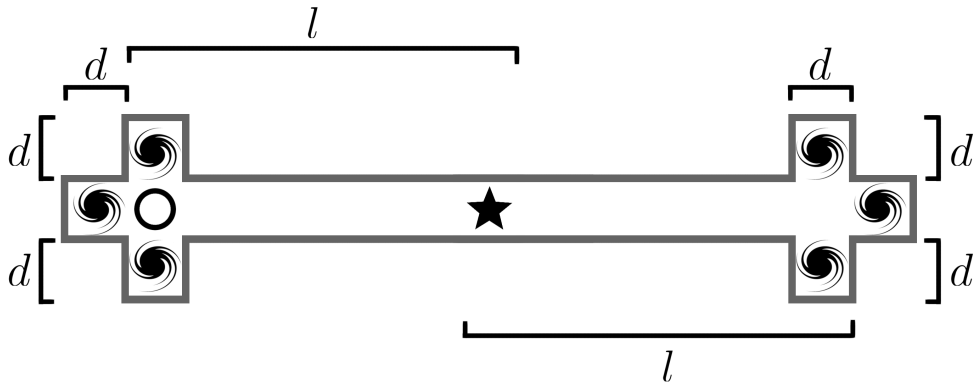

Abbildung 5.6: Knoten-Gadget bei unidirektionalen Wurmlöchern. Die Wurmlöcher auf der linken Seite sind ankommende, die auf der rechten Seite abgehende Wurmlöcher.

5.2 Theorem 2: Sammelobjekte und Stoppuhr

Für unser nächstes Theorem führen wir das Spielelement *Sammelobjekt* ein. Sammelobjekte sind Gegenstände, die in einem Level an verschiedenen Orten platziert sind und von dem Spieler eingesammelt werden können. Dabei können an einer Position beliebig viele Sammelobjekte vorhanden sein. Ein Ziel des Spiels ist es, möglichst viele Sammelobjekte in einem Level einzusammeln. Dies entspricht auf natürliche Weise dem Bestreben, ein Spiel möglichst gut abzuschließen. Wir werden dieses Element mit dem bereits oben beschriebenen Element der *Stoppuhr* kombinieren. Dies bedeutet, dass der Spieler nur begrenzt Zeit hat, ein Level zu beenden. Des Weiteren fordern wir, dass alle Wege *uniform* sind. Der Avatar bewegt sich somit in jede Richtung gleich schnell. Zuletzt muss der Spieler mit seinem Avatar eine ausgezeichnete Zielposition innerhalb der durch die Stoppuhr gesetzten Zeit erreichen, um das Level erfolgreich abzuschließen. Bei der beschriebenen Aufgabe handelt es sich also um ein Optimierungsproblem. Betrachten wir die Aufgabe als Entscheidungsproblem, so lautet die Fragestellung:

Definition 5 (Entscheidungsproblem 2)**.**

Eingabe:
Ein Level mit n Positionen, an denen $a_1, \ldots, a_n \in \mathbb{N}$ Sammelobjekte platziert sind, einer Startposition u und einer Zielposition v, sowie eine Stoppuhr mit Zeit t und eine Zahl $k \in \mathbb{N}$.
Frage:
Ist es möglich, innerhalb der Zeit t von der Startposition u aus mindestens k Sammelobjekte einzusammeln und die Zielposition v zu erreichen?

5.2.1 Anmerkungen

Ein ähnliches Theorem wurde bereits 2010 von Michal Forišek [For10] bewiesen. Sein Metatheorem lautet:
„*A 2D platform game where the collecting items feature is present and a time limit is present as a part of the instance is NP-hard,*“
wobei das *collecting items feature* seiner Beschreibung nach unseren zwingenden Lokationen entspricht. Michal Forišek führt den Beweis dieses Theorems über eine Reduktion von dem NP-vollständigen Hamilton Cycles on Grid Graphs Problem [IPS82]. Dabei wird in jeden Knoten des Graphen ein *collectable item* platziert und das *time limit* wird auf $(|V| - 1) \cdot \alpha$ gesetzt, wobei $|V|$ die Anzahl der Knoten des Graphen bezeichnet und der konstante Faktor α die Zeit beschreibt, die zum Überqueren einer Kante benötigt wird. Hierbei wird implizit davon ausgegangen, dass sich der Avatar in jede Richtung gleich schnell bewegt. Diese Einschränkung ist für die Reduktion von Hamilton Cycles on Grid Graphs zwingend erforderlich, da nicht vorher gesagt werden kann, wie viele Kanten pro Bewegungsrichtung passiert werden. Unterschiedliche Geschwindigkeiten pro Bewegungsrichtung führen somit dazu, dass das *time limit* nicht mehr genau berechnet werden kann. Setzt man die schnellste Bewegungsrichtung voraus, so kommt es durch die langsameren Richtungen zu Verzögerungen, sodass das Zeitlimit selbst bei vorhandenem Hamiltonkreis nicht eingehalten werden kann. Setzt man hingegen eine langsame Bewegungsrichtung voraus, so kann es bei großen Eingaben vorkommen, dass durch Abkürzungen entlang der schnelleren Bewegungsrichtung so viel Zeit eingespart werden kann, dass Knoten mehrfach besucht werden können und somit alle zwingenden Lokationen erreichbar sind, ohne dass ein Hamiltonkreis existiert. Michal Forišek müsste in seinem Theorem somit neben *collectable items* und einem *time limit* auch *uniforme Wege* fordern. In dem von uns im Folgenden angegebenen Theorem können wir auch ohne uniforme Wege auskommen. Da es jedoch die Konstruktion vereinfacht, gehen wir zunächst von uniformen Wegen aus und zeigen in einem späteren Korollar, dass wir auf diese Forderung verzichten können.
Wie in dem vorangegangenen Theorem wollen wir das Spiel wieder dahingehend verallgemeinern, dass wir beliebig große Level zulassen. Dieses Mal müssen wir uns jedoch Gedanken um die Kodierung des Spiels machen. Angelehnt an „Super Mario Bros. Is Harder/Easier than We Thought“ [DVW16] verwenden wir eine sogenannte *Run-Length-Encoding* (kurz RLE). Hierbei wird eine Reihe von k identischen Elementen durch eine einzelne Kopie des Elements und der Binärdarstellung von k gespeichert. Wir können somit exponentiell lange identisch aufgebaute Wegstücke auf polynomiellen Platz speichern. In dem angegebenen Artikel wird die RLE verwendet, um zu zeigen, dass „Super Mario Bros.“ unter Verwendung der RLE schwach-NP-schwer ist, selbst wenn nur ein konstant großes Fenster des Spielfelds gleichzeitig

zu sehen ist und der Zustand aller Elemente außerhalb dieses Fensters nicht gespeichert wird (beispielsweise erscheinen Gegner, die einmal besiegt wurden, bei nochmaligem Besuch ihrer Position wieder, jedoch wird gefordert, dass sich das Fenster nur nach rechts verschieben kann). Hierbei wird der Beweis durch eine Reduktion des schwach-NP-vollständigen KNAPSACK Problems [Sch10] geführt, wobei der für „Super Mario Bros." übliche Timer benutzt wird, um die Gewichtsschranke des Knapsacks zu realisieren. Das von uns im Folgenden vorgestellte Theorem verallgemeinert diesen Ansatz und greift dabei nicht auf spezielle, dem Spiel eigene Elemente, wie zum Beispiel die Röhren, zurück. Unser Theorem ist unabhängig und ohne Kenntnis von der Arbeit „Super Mario Bros. Is Harder/Easier than We Thought" [DVW16] entstanden. Erst bei der Diskussion der Notwendigkeit der RLE sind wir auf den Artikel aufmerksam geworden.

Fassen wir Spiele gemäß dem ENTSCHEIDUNGSPROBLEM 2 (Def. 5) auf, so können wir folgendes Theorem formulieren:

Theorem 2.

Ein Spiel, das Sammelobjekte und eine Stoppuhr enthält, dessen Wege uniform sind und bei dem eine Zielposition erreicht werden muss, ist schwach-NP-schwer.[4]

Beweis:

Wir führen den Beweis durch eine Reduktion von dem schwach-NP-vollständigen KNAPSACK Problem. Das KNAPSACK Problem ist wie folgt definiert [GJ79]:

Definition 6 (KNAPSACK Problem)**.**

Eingabe:
Eine endliche Menge Q, ein Mindestnutzen $N \in \mathbb{Z}^+$, eine Kostenschranke $W \in \mathbb{Z}^+$, sowie für jedes $q_i \in Q$ einen Nutzwert $a_i \in \mathbb{Z}^+$ und Kosten $b_i \in \mathbb{Z}^+$.
Frage:
Existiert eine Teilmenge $Q' \subseteq Q$ mit $\sum_{q_i \in Q'} a_i \geq N$ und $\sum_{q_i \in Q'} b_i \leq W$?

Reduktion

Die Idee hinter unserer Reduktion ist, die Objekte der Knapsack-Instanz auf Positionen mit Sammelobjekten abzubilden. Die Anzahl der Sammelobjekte pro Position entspricht dabei dem Nutzwert des Objekts. Das Level ist

[4] Hierbei betrachten wir Spiele gemäß dem ENTSCHEIDUNGSPROBLEM 2

wie in Abbildung 5.7 aufgebaut. Die roten Punkte stellen Positionen mit Sammelobjekten und die schwarzen Punkte stellen die Startposition u und die Zielposition v dar. Die roten Punkte sind jeweils über unterschiedlich lange Pfade mit einem horizontalen Pfad, der u und v verbindet (im Folgenden Hauptpfad genannt), verbunden. Der Abstand von einem roten Punkt zum Hauptpfad entspricht genau dem Gewicht des Objekts, das dieser rote Punkt repräsentiert. Die Gewichtsschranke der Knapsack-Instanz wird auf die Stoppuhr der Spiel-Instanz und der Mindestnutzen auf die Zahl k abgebildet. Dabei wird es genau dann in dem Level möglich sein, innerhalb der durch die Stoppuhr vorgegebenen Zeit, mindestens k Sammelobjekte einzusammeln, wenn es in der Knapsack-Instanz möglich ist, innerhalb der Kostenschranke W einen Nutzwert $\geq N$ zu erreichen.

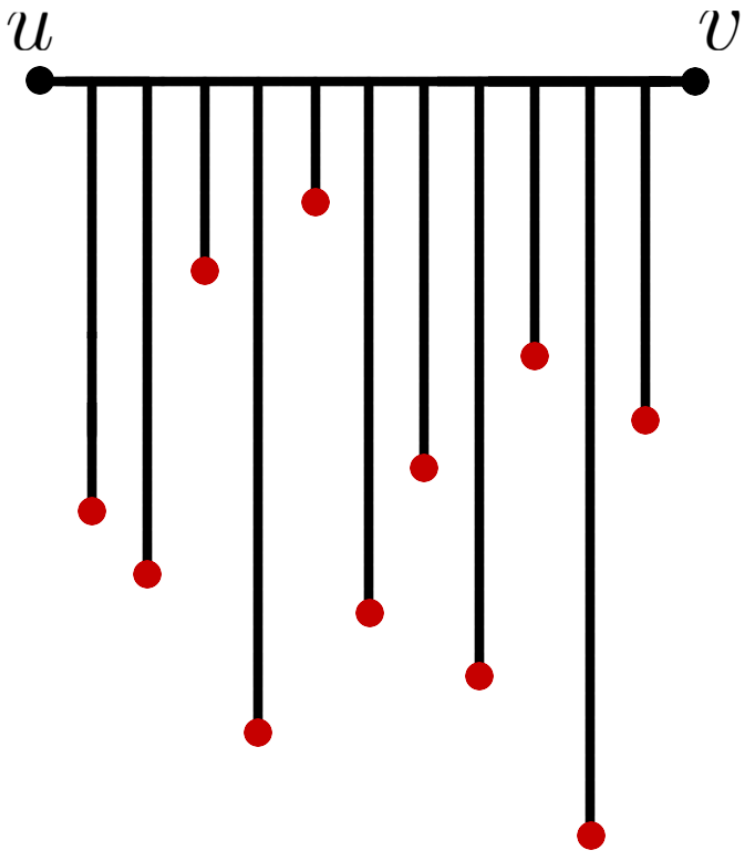

Abbildung 5.7: Konstruiertes Level.

Konstruktion

Zunächst überprüfen wir, ob es sich bei der Eingabe w um eine für das KNAPSACK-Problem korrekt kodierte Eingabe handelt. Ist dies nicht der Fall, so bilden wir die Instanz auf ein Level ab, das nicht gewinnbar ist (zum Beispiel, weil es keinen Pfad zwischen Start- und Zielposition gibt). Wir gehen also im Folgenden davon aus, dass es sich bei w um eine korrekt kodierte Eingabe handelt und wir möchten nun, in Abhängigkeit von w, eine Spielinstanz generieren.
Um das konstruierte Level leichter beschreiben zu können, legen wir ein zweidimensionales Koordinatensystem über das Level. Die x-Achse verläuft dabei nach rechts und die y-Achse nach unten.
Zunächst benötigen wir eine Start- und eine Zielposition u und v. Die Startposition u wird auf dem Ursprung des Koordinatensystems platziert und wenn w gerade n Objekte beinhaltet, so erhält v die Koordinaten $(n+1, 0)$. Wir verbinden u und v mit einem Pfad, welchen wir Hauptpfad nennen. Für jedes der n Objekte $q_1, \ldots, q_n$ erzeugen wir eine Position, die in Abbildung 5.7 durch einen roten Punkt dargestellt wird. Hat q_i die Kosten b_i und den Nutzwert a_i, so wird an den Koordinaten (i, b_i) eine Lokation mit a_i Sammelobjekten erstellt und über einen Pfad der Länge b_i vertikal mit dem Hauptpfad verbunden. Platzieren wir die Lokationen jeweils in einem x-Abstand von Eins zu einander, so beträgt die Länge des Hauptpfades $d = n+1$. Der Term d kommt daher, dass wir nicht alle roten Punkte so verbinden können, dass sich die Verbindungspfade an demselben Punkt treffen, da die meisten Spiele nur vier Bewegungsrichtungen zulassen. Da wir somit gezwungen sind, den Hauptpfad einzuführen, um die Verbindungspfade miteinander zu verbinden, kommt es zu Verlängerungen der Pfade, nämlich genau um den Weg, den der Avatar über den Hauptpfad zurücklegen muss. Da der Avatar jedoch in jedem Fall einmal den Hauptpfad überqueren muss, um zur Zielposition zu gelangen, können wir einfach dem Wert der Stoppuhr einen Term für die Überquerung des Hauptpfades addieren. Da der Spieler mit seinem Avatar die Zielposition erreichen muss, um das Level abzuschließen, ist er ebenfalls gezwungen, jede durch einen roten Punkt markierte Position nach dem Einsammeln der Sammelobjekte über denselben Pfad, über den er die Position erreicht hat, wieder zu verlassen. Somit wurde jedem Objekt q_i mit Gewicht b_i ein Wegstück der Länge $2b_i$ zugewiesen. Die Gewichte wurden somit um den Faktor 2 gestreckt. Dies führt uns zu einer Gesamtkostenschranke für den erlaubten Weg von $2 \cdot W + d$. Sei δ die für ein Wegstück der Länge Eins benötigte Zeit. Da die Wege nach Voraussetzung uniform sind, ist die Richtung hierbei beliebig. Die Stoppuhr wird damit auf $(2 \cdot W + d) \cdot \delta$ gesetzt. Da wir den Nutzwert pro Objekt direkt durch die dem Nutzwert entsprechende Anzahl an Sammelobjekten abbilden, können wir $k := N$ auf den Mindestnutzen setzen.

Korrektheitsbeweis

Wir wollen zeigen, dass es genau dann möglich ist, den Rucksack innerhalb der Kostenschranke W so zu befüllen, dass ein Mindestnutzen N erreicht wird, wenn es möglich ist, das konstruierte Level innerhalb der durch die Stoppuhr vorgegebenen Zeit abzuschließen und dabei mindestens k Sammelobjekte einzusammeln.
Wir haben bei der Konstruktion darauf geachtet, dass die Kostenschranke W um den gleichen Faktor, wie die Gewichte $b_1, \ldots, b_n$ der Objekte, gestreckt wird. Zusätzlich wurde zu W der Wert d addiert. Dieser Puffer wird aber in jedem Fall auf dem Hauptpfad verbraucht, da wir immer von der Startposition zur Zielposition gelangen müssen. Für das Einsammeln der Sammelobjekte steht also nur ein Streckenanteil der Länge $2 \cdot W$ zur Verfügung.

Durch die Änderung der Kostenschranke können somit keine zusätzlichen Objekte zur Lösung hinzugefügt werden, die nicht Teil der ursprünglichen Lösung sind. Die Tatsache, dass die Repräsentationen aller Objekte die zur ursprünglichen Lösung gehören, ebenfalls zur Lösung der Spiel-Instanz gehören, folgt unmittelbar aus der Konstruktion, da wir darauf geachtet haben, die Zeit der Stoppuhr so zu setzen, dass die zur Verfügung stehende Zeit ausreicht, um einen Weg abzulaufen, der alle Positionen enthält, deren abgebildete Objekte Teil der Lösung der Knapsack-Instanz sind.
Somit haben wir gezeigt, dass $w \in$ Knapsack-Problem ist, genau dann, wenn das konstruierte Level unter den geforderten Bedingungen lösbar ist.

Laufzeit

Wir müssen $n + 2$ Positionen erstellen. Diese Positionen müssen wir mit Pfaden der Gesamtlänge $\sum_{i=1}^{n} b_i + d$ verbinden. Da wir Run-Length-Encoding benutzen, benötigen wir nur polynomiellen Platz und polynomielle Zeit um das entsprechende Level zu kodieren. □

5.2.2 Uniforme Wege

Wir können die Forderung der uniformen Wege auch weglassen. Dies wird dadurch möglich, dass wir von unserer Konstruktion her wissen, welchen Wegabschnitt wir in welche Richtung passieren werden. Betrachten wir Spiele gemäß dem Entscheidungsproblem 2, so erhalten wir folgendes Korollar:

Korollar 1.

Ein Spiel, das Sammelobjekte und eine Stoppuhr enthält und bei dem eine Zielposition erreicht werden muss, ist schwach-NP-schwer[5].

Beweis:

Die uniformen Wege forderten, dass sich der Avatar in jede Richtung gleich schnell bewegt. Verzichten wir auf diese Forderung, so müssen wir statt einem Umrechnungsfaktor δ nun mehrere Faktoren betrachten. Da unser Level so konstruiert ist, dass wir drei Bewegungsrichtungen (hoch, runter und nach rechts) verwenden, werden wir drei unterschiedliche Bewegungsgeschwindigkeiten betrachten. Die Konstruktion für eine andere Anzahl an Bewegungsrichtungen erfolgt ähnlich.
Wir führen den Beweis analog zu Theorem 2. Lediglich der Wert für die Stoppuhr wird den verschiedenen Bewegungsrichtungen angepasst. Bei Theorem 2 haben wir die Stoppuhr auf $(2 \cdot W + d) \cdot \delta$ gesetzt. Dieser Wert setzt sich zusammen aus $W \cdot \delta$ um die Korridore mit den Sammelobjekten am Ende hinunter zu laufen, $W \cdot \delta$ um die Korridore mit den Sammelobjekten wieder hoch zu laufen und $d \cdot \delta$ um den Hauptpfad von der Startposition aus nach rechts zur Zielposition zu durchqueren. Verwenden wir nun statt einem Faktor δ die Faktoren $\delta_o, \delta_u, \delta_r$ für die Zeit, die der Avatar mindestens benötigt, um ein Wegstück der Länge Eins nach oben, unten und rechts zu durchqueren, so erhalten wir als neuen Wert für die Stoppuhr $W \cdot \delta_u + W \cdot \delta_o + d \cdot \delta_r$. Wir wissen, dass wir in jedem Fall den Term $d \cdot \delta_r$ auf die Durchquerung des Hauptpfads von links nach rechts verwenden müssen. Da wir eine Zielposition erreichen müssen, müssen wir jeden Korridor, den wir nach unten passiert haben, auch wieder nach oben durchqueren, um zum Ziel gelangen zu können. Also können wir keine zeitlichen Einsparungen in eine Richtung erreichen, da jeder Pfad nach unten an einen Pfad nach oben gekoppelt ist. Aus unserer Konstruktion und aus der Diskussion von Theorem 2 folgt sofort, dass $w \in$ Knapsack-Problem ist, genau dann wenn das konstruierte Level unter den geforderten Bedingungen lösbar ist. □

5.2.3 Technische Details

In obigem Korollar sind wir davon ausgegangen, dass der Avatar, sobald er die Bewegung startet, sofort die maximale Geschwindigkeit in eine Richtung erreicht. Wir gehen also davon aus, dass es keine Beschleunigungsphasen gibt. Diese Einschränkung ist allerdings nicht nötig, da wir Beschleunigungsphasen endlicher Dauer erlauben können, indem wir die Korridore der Sammelobjekte soweit strecken, dass der Zeitunterschied, der durch die Beschleunigungsphasen auftritt, im Vergleich zur benötigten Zeit für die Gesamtstrecke vernachlässigbar klein wird. Dieser Unterschied ist für ein einzelnes Auftreten

einer Beschleunigungsphase konstant. Da wir n Abzweigungen vom Hauptpfad haben, können höchstens $\mathcal{O}(n)$ Beschleunigungsphasen pro Level auftreten. Somit ist der Unterschied der benötigten Zeit durch die Beschleunigungsphasen linear in der Eingabe beschränkt und wir können für ein konkretes Spiel einen Faktor c finden, sodass der Zeitverlust durch die Beschleunigungsphasen pro Level höchstens $c \cdot n$ beträgt. Wir fügen dem Wert für die Stoppuhr den Summanden $c \cdot n$ hinzu und strecken die Korridore der Sammelobjekte um den Faktor $c \cdot n$. Nun ist sicher gestellt, dass wir schon bei dem Besuch des kürzesten Korridors den hinzugefügten Puffer komplett aufbrauchen und somit keine zusätzliche Sammelobjekt-Position in die Lösung mit aufnehmen können, die zuvor nicht Teil der Lösung war.
Spezielle, spielabhängige Manöver, die die Geschwindigkeit eines Avatars in eine Richtung ändern, wie zum Beispiel Springen oder Rutschen, können ebenfalls vernachlässigt werden, da die Level entweder so konstruiert werden können, dass diese Manöver nicht möglich sind (zum Beispiel durch niedrige Korridore, in denen Springen nicht möglich ist), oder diese Manöver mit einbezogen werden. Hierbei gehen wir davon aus, dass sie so oft wie möglich angewendet werden und betrachten die schnellst mögliche Geschwindigkeit in eine Richtung.
Bei der Berechnung des Wertes der Stoppuhr kann es vorkommen, dass diese einen Wert aus $\mathbb{R}^+$ erhält. Fordert das Spiel jedoch Zeit-Werte aus $\mathbb{N}$, so kann der Abschnitt des Hauptpfades nach der letzten Abzweigung bis zur Zielposition so weit gestreckt werden, dass ein benötigter Timer-Wert aus $\mathbb{N}$ erreicht wird.

5.2.4 Diskussion

Da wir bei dem zweiten Theorem gezwungen waren wie in dem Artikel „Super Mario Bros. Is Harder/Easier than We Thought“ [DVW16] die Run-Length-Encoding zu verwenden, konnten wir aufgrund des Pseudopolynomzeitalgorithmuses für Knapsack nur schwache NP-Schwere zeigen. Es ist jedoch fraglich, inwiefern dieses Ergebnis für Spiele relevant ist, da das Lösen eines Levels in natürlicher Form aus dem schrittweisen Durchspielen des Levels besteht und nicht aus Zusammenfassungen von Abschnitten. Ein schrittweises Durchspielen würde bei unserem konstruierten Level eine exponentielle Laufzeit bedeuten. Da aber nach dem Artikel [DVW16] die RLE auch in konkreten Spielen implementiert ist, denken wir, dass unser Ergebnis trotzdem hier vorgestellt werden sollte.
Wir möchten an dieser Stelle darauf hinweisen, dass Theorem 1 ohne RLE auskommt. Hier können die konstruierten Level unkomprimiert kodiert werden, da wir linear in der Eingabe viele Gadgets konstanter Größe erzeugen. Theorem 1 benötigt auch nicht die Forderung der uniformen Wege. Wir gehen

zwar in der Konstruktion davon aus, dass der Avatar sich nach rechts und links gleich schnell bewegt, können jedoch auf diese Forderung verzichten, da wir wissen, dass der Avatar ein Knoten-Gadget genau einmal in der Richtung von links nach rechts und einmal in der Richtung von rechts nach links durchläuft. Ähnlich wie in Korollar 1 können wir nun die Gesamtwegstrecke aufteilen in eine Wegstrecke, die von links nach rechts und eine, die von rechts nach links durchlaufen wird und die Strecken entsprechend mit eigenen Zeitfaktoren gewichten.

Kapitel 6

Anwendung

Beide Theoreme aus dem vorangegangenen Abschnitt lassen sich auf das Spiel „Super Mario Bros.“ anwenden. Wir wollen nun zunächst das Spiel vorstellen und anschließend auf vorherige komplexitätstheoretische Untersuchungen des Spiels eingehen. Zur Untersuchung des Spiels wurde eine online emulierte Kopie des Spiels betrachtet, welche unter `http://www.playretrogames.com/3171-super-mario-bros` [Mar14] zu finden ist.

6.1 „Super Mario Bros.“ - Das Spiel

„Super Mario Bros.“ ist ein 2D Jump&Run Spiel, das im Jahre 1987 erstmals im europäischen Raum für das Nintendo Entertainment System veröffentlicht wurde [Sch14]. In den darauffolgenden Jahren wurden etliche Nachfolger dieses Spiels veröffentlicht und die Hauptfigur Mario wurde auf der ganzen Welt berühmt. Ziel des Spiels ist es, Mario durch verschiedene Level zu führen, um letztlich die entführte Prinzessin Peach zu retten. Die Level sind dabei voller Gefahren, wie todbringende Abgründe, Lava-Felder und rotierende Feuer-Ketten, Gegner, von denen manche besiegt werden können, indem Mario auf sie springt, und andere nicht und vieles mehr. Mario hat hierbei nur die Möglichkeit, nach links und rechts zu gehen und zu springen. Da das Spiel Schwerkraft simuliert, fällt Mario nach unten, sobald er keinen Block mehr unter sich hat. Der Ausgang eines Levels, und somit dessen Ziel, ist mit einer Flagge markiert. Ein Level ist aus Blöcken verschiedener Sorten aufgebaut. Bei den meisten Blöcken handelt es sich um unzerstörbare Blöcke. Sie bilden das Grundgerüst des Levels. Neben diesen Blöcken gibt es noch zerstörbare Blöcke, die Mario in seiner großen Form durch Dagegenspringen von unten zerstören kann. Mario beginnt das Spiel in seiner kleinen Form und wird, sobald er einen Wachstumspilz konsumiert hat, zum großen Super

Mario. Wird der große Mario von einem Gegner getroffen, so wird er wieder zu dem kleinen Mario. Wird hingegen der kleine Mario von einem Gegner getroffen, so stirbt er. Abbildung 6.1 zeigt Mario links in seiner kleinen Form und rechts in seiner großen Form.

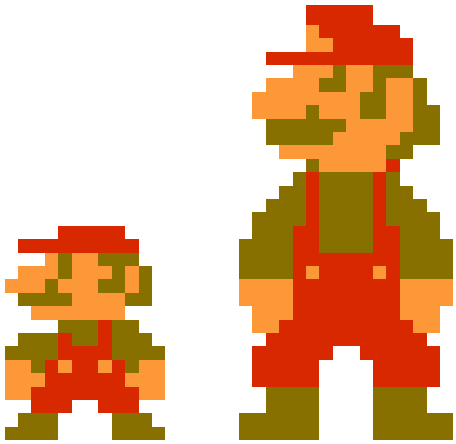

Abbildung 6.1: Links ist der kleine Mario und rechts der große Super Mario zu sehen [Mar04].

Eine besondere Blockart bilden die Fragezeichenblöcke, welche unterschiedliche Items freigeben, wenn Mario von unten dagegen springt. Diese Items umfassen einsammelbare Münzen, Wachstumspilze, Feuerblumen, welche Mario Feuerbälle verschießen lassen, Unbesiegbarkeits-Sterne und einen 1-Up-Pilz, welcher Mario ein zusätzliches Leben verleiht. Mario können mehrere Leben zur Verfügung stehen und jedes Mal, wenn er durch einen Gegner besiegt wurde, oder durch eine Falle zu Tode kam, verliert er ein Leben und muss das Level von dem letzten Checkpoint ab neu beginnen. Hat Mario alle Leben verloren, so ist das ganze Spiel verloren.
Am rechten oberen Bildschirmrand ist ein Timer zu sehen, der anzeigt, wie viel Zeit dem Spieler noch bleibt, um das Level abzuschließen. Läuft der Timer ab, bevor Mario die Ziel-Flagge erreicht hat, so verliert er ein Leben und muss beim letzten Checkpoint von vorne beginnen.
Ein weiteres interessantes Element des Spiels sind die Röhren. Diese können simple Plattformen sein, aus ihnen können periodisch gefährliche Pflanzen emporsteigen, welche Mario bei Berührung verletzen, oder sie können Mario als Abkürzung dienen oder ihn in versteckte Bereiche des Levels führen.

6.2 Bisherige Arbeiten zu „Super Mario Bros.“

„Super Mario Bros.“ wurde schon mehrfach auf seine Komplexität untersucht. In dem 2015 erschienenen Paper „Classic Nintendo Games are (NP-)Hard“ [ADGV15] wird eine Reduktion von dem NP-vollständigen SAT Problem [GJ79] geführt. Hierbei wird jedoch nicht auf den Timer eingegangen, der in „Super Mario Bros.“ vorhanden ist und den Spieler zwingt, ein Level innerhalb einer bestimmten Zeit abzuschließen. Da in dem Artikel von beliebig großen Leveln ausgegangen wird, müsste auch der Timer so gesetzt werden, dass diese möglicherweise sehr großen Level innerhalb des Timers gelöst werden können. Da die Kodierung des Spiels bei einer Reduktion von SAT nicht zu Problemen führt, wird in dem Artikel nicht auf die betrachtete Kodierung eingegangen. Ein Jahr später wird in dem Paper „Super Mario Bros. Is Harder/Easier than We Thought“ [DVW16] das Spiel noch einmal genauer untersucht. Es werden verschiedene Kodierungen und Fenstergrößen betrachtet. Die Fenstergröße bezeichnet den Bereich des Spielfelds, der auf einmal zu sehen ist. In vorherigen Untersuchungen des Spiels wurde immer davon ausgegangen, dass das gesamte Spielfeld gleichzeitig zu sehen ist, und der Zustand aller Objekte im Spiel immer gespeichert wird. In dem originalen „Super Mario Bros.“ Spiel werden dem Artikel „Super Mario Bros. Is Harder/Easier than We Thought“ [DVW16] nach die Zustände aller Objekte, welche sich nicht innerhalb des sichtbaren Fensters befinden, wieder vergessen. Dies bedeutet insbesondere, dass Gegner, die schon einmal besiegt wurden, bei erneutem Besuch ihrer Position wieder lebendig sind. In dem Artikel wurde zudem die Eigenschaft, dass Mario mehrere Leben hat und durch bestimmte Handlungen neue Leben hinzu bekommen kann, betrachtet. Außerdem wurde neben dem Spielziel, ein einzelnes Level zu gewinnen, auch das Ziel, eine Reihe von Leveln zu gewinnen und die Prinzessin zu befreien[1], betrachtet. Man kam zu dem Ergebnis, dass die Frage, ob ein Level gewinnbar ist, in „Super Mario Bros.“ ohne Run-Length-Encoding (kurz RLE) mit konstanter Fenstergröße in deterministischer Polynomzeit entscheidbar ist. Ebenfalls ist die Frage, ob man die Prinzessin retten kann, unter diesen Bedingungen in P. Verwendet man die RLE und fordert, dass man in dem Level nicht zurück gehen kann (das Fenster bewegt sich nur nach rechts, nicht nach links), so wird die Frage, ob man die Prinzessin retten kann, schwach-NP-schwer. Verzichtet man auf RLE und fordert, dass der Zustand aller Objekte gespeichert wird, unabhängig davon, ob sie sich innerhalb des Fensters befinden, so ist die Frage, ob ein Level lösbar ist, PSPACE-vollständig.
In dem NP-Schwere Beweis wird insbesondere der Timer verwendet, um eine

[1] Die Motivation hinter dem Spiel ist es, Prinzessin Peach zu befreien, die von dem bösen König Koopa, oder später Bowser genannt, entführt wurde [Mar85].

Reduktion von KNAPSACK, ähnlich der unsrigen zu Theorem 2, zu führen.
Da diese Reduktion ähnlich zu unserer verlaufen würde, verzichten wir an dieser Stelle darauf, Theorem 2 auf „Super Mario Bros.“ anzuwenden und verweisen auf den Artikel „Super Mario Bros. Is Harder/Easier than We Thought“ [DVW16]. Wir wollen stattdessen Theorem 1 auf „Super Mario Bros.“ anwenden. Wir verzichten hierbei auf RLE, fordern aber, dass die Zustände aller Objekte außerhalb des Sichtfensters gespeichert werden und gehen davon aus, dass die Benutzung der Röhren diese Zustände ebenfalls nicht zurücksetzt.

6.3 Realisierung der Spielelemente

Um unser Theorem auf „Super Mario Bros.“ anwenden zu können, müssen wir zeigen, dass sich zwingende Lokationen, Wurmlöcher und eine Stoppuhr mit den durch das Spiel bereitgestellten Elementen realisieren lassen. Die Stoppuhr ist bereits im Spiel vorhanden und kann genau so übernommen werden. Die Wurmlöcher können durch Röhren realisiert werden. Die Röhren verhalten sich in „Super Mario Bros.“ unterschiedlich. Bei der Benutzung mancher Röhren fällt Mario an einer anderen Position im Level vom oberen Bildschirmrand herunter, bei anderen Röhren steigt er nach der Benutzung der Röhre aus einer anderen Röhre an einer anderen Position wieder heraus. Wir gehen davon aus, dass alle Röhren das zweit genannte Verhalten zeigen.
Die zwingenden Lokationen realisieren wir durch Hindernisse, die nur überwunden werden können, wenn Mario direkt davor einen Stern konsumiert, sodass er unbesiegbar wird. Die Blöcke, in denen sich der Stern befindet, können von den Knoten-Gadgets aus aktiviert werden und dienen somit als zwingende Lokationen. Wurde ein Stern aktiviert, so bleibt er wegen den Blöcken links und rechts von dem Fragezeichenblock an Ort und Stelle und kann später von Mario eingesammelt werden. Die Hindernisse werden der Reihe nach vor dem Ziel platziert, sodass der Spieler gezwungen ist, diese zu passieren, um das Level abzuschließen. Da er nach dem Passieren des ersten Hindernisses nicht mehr zurück kann, kann er die Strecke mit den Hindernissen erst überqueren, wenn er alle Sterne aktiviert hat. Das beschriebene Gadget ist von dem Clause-Gadget aus dem Artikel „Classic Nintendo Games are (Computationally) Hard“ [ADGV15] inspiriert. Wir platzieren nun noch an der Startposition einen Wachstumspilz, da in „Super Mario Bros.“ nur der große Mario in der Lage ist, andere Items, als den Wachstumspilz, aus Fragezeichenblöcken zu erhalten.
Die Röhren können zusätzlich noch mit Zahlen versehen werden, um zu verdeutlichen, welche Röhren verbunden sind. Auch in „Super Mario Bros.“ sind bestimmte Röhren mit Zahlen versehen, um zu verdeutlichen, wohin sie führen.

Die beschriebene Konstruktion ist in Abbildung 6.2 zu sehen. Dabei sind die Sterne und der Wachstumspilz, welche in den Fragezeichenblöcken stecken, zur Veranschaulichung abgebildet, während sie in dem späteren Level erst sichtbar werden, wenn Mario die Blöcke aktiviert hat. Die Länge der Hindernisse muss hierbei so gewählt werden, dass Mario sie nur überwinden kann, wenn er einen Stern konsumiert hat. Dabei ist zu beachten, dass der große Mario bei Kontakt mit einem Hindernis oder Gegner ebenfalls für kurze Zeit unverwundbar ist, während er zu dem kleinen Mario wird. Sollte dies zu Problemen führen, so kann ein kürzeres Hindernis ohne Stern vor jedem Hindernis platziert werden, durch welches Mario in seine kleine Form gezwungen wird. Nach dem eigentlichen Hindernis wird dann ein weiterer Wachstumspilz platziert, sodass Mario bei dem nächsten Hindernis in seiner großen Form startet. Gleichzeitig müssen die Hindernisse lang genug sein, damit Mario mit einem Stern nicht zwei Hindernisse durchqueren kann.

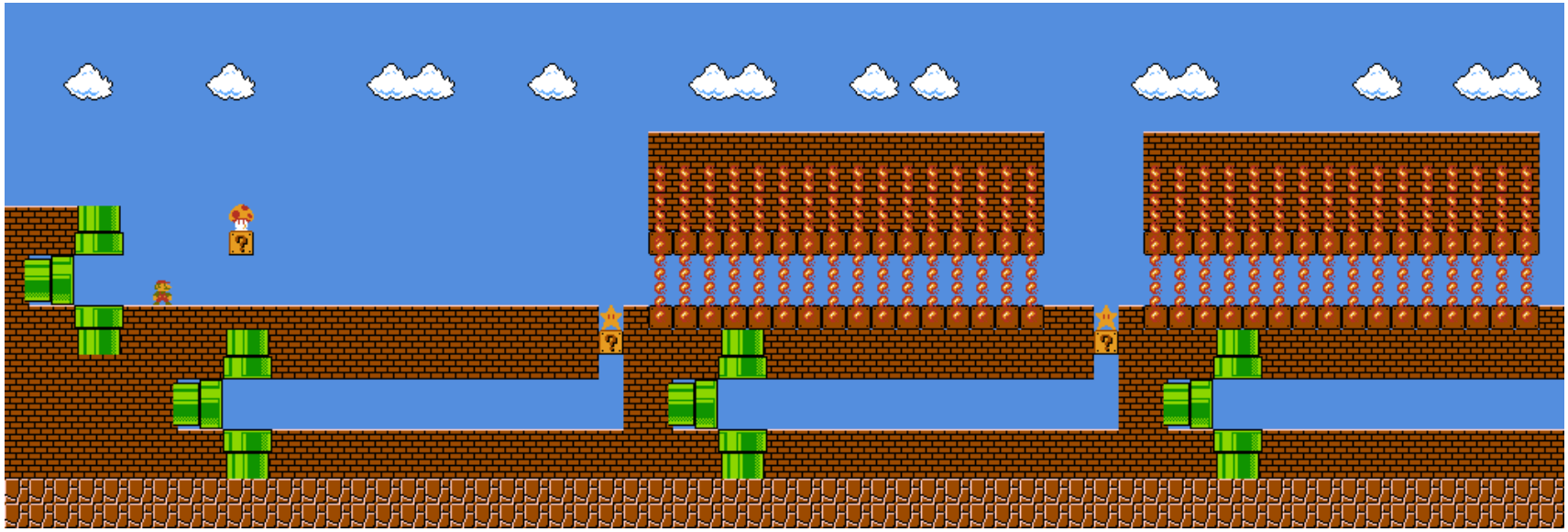

Abbildung 6.2: Mario befindet sich an der Startposition. Er kann nun die Röhren links von ihm benutzen, um in den unteren Teil des Levels zu gelangen und die Sterne zu aktivieren [Mar15].

Es ist uns somit gelungen, alle benötigten Spielelemente zu realisieren. Der folgende Satz ist somit eine Konsequenz von Theorem 1.

Satz 2.

Das Spiel „Super Mario Bros.“ in der beschriebenen verallgemeinerten Form ist NP-schwer.

Beweis:

Alle Spielelemente aus Theorem 1 konnten mit den Bausteinen, die das Spiel zur Verfügung stellt, realisiert werden. Nach Theorem 1 ist das Spiel somit NP-schwer. □

Kapitel 7

Diskussion und Ausblick

Zuletzt möchten wir noch einmal die vorliegende Arbeit zusammenfassen und Ansätze für eine weiterführende Forschung auf diesem Gebiet geben.
Zu Beginn haben wir die Arbeit von Giovanni Viglietta vorgestellt und eines seiner Metatheoreme auf das Spiel „Super Mario World 2: Yoshi's Island“ angewandt und damit dessen bisher unbekannte NP-Schwere gezeigt. Dazu haben wir die in dem Metatheorem geforderten Spielelemente mit Bausteinen, die das Spiel zur Verfügung stellt, realisiert.
Als nächstes haben wir das Spielelement der Zeit betrachtet und vorangegangene Arbeiten, die sich mit diesem Element beschäftigt haben, diskutiert. Anschließend haben wir zwei selbst entwickelte Metatheoreme vorgestellt, die sich mit diesem Element beschäftigen. Dabei haben wir die Forderung uniformer Wege diskutiert, welche in bisherigen Ergebnissen unbeachtet blieb. Zudem haben wir unterschieden, wie ein Spiel kodiert wird. Während wir bei unserem ersten Theorem davon ausgingen, dass ein Level diskret ist und jede Position in dem Level direkt gespeichert wird, gingen wir in dem zweiten Theorem davon aus, dass Wegstücke gleicher Art zusammengefasst werden und nur ein elementares Wegstück, gefolgt von einer Binärzahl, die die Anzahl der Wiederholungen in eine Richtung angibt, gespeichert wird. Abschließend wurden noch einige technische Details der Konstruktionen diskutiert. Im letzten Abschnitt dieser Arbeit wurde das Spiel „Super Mario Bros.“ betrachtet. Dazu wurden ebenfalls zunächst die bisherigen Arbeiten zu diesem Spiel vorgestellt und anschließend wurde unser erstes Theorem auf das Spiel angewandt, indem wieder alle geforderten Elemente mit Bausteinen des Spiels realisiert wurden. Dabei berücksichtigte unser Beweis der NP-Schwere von „Super Mario Bros.“ das Element der Zeit, während dieses in vorherigen Arbeiten zum Teil ausgelassen wurde.
Als Hauptergebnis dieser Arbeit kann man die beiden Metatheoreme ansehen, durch die nun zwei weitere Werkzeuge zur Bestimmung der Komplexität von Computerspielen zur Verfügung stehen.

Ich hoffe, dass die Forschung auf diesem Gebiet sich weiter dahin entwickelt, Metatheoreme und Frameworks zu erforschen, die die Komplexitätsbestimmung mehrerer Spiele auf einmal ermöglichen. Gerade durch die fortwährende Entwicklung der Hardware über die letzten Jahrzehnte ist es möglich geworden, immer mehr und komplexere Spiele zu entwickeln, sodass der Ansatz konkrete Spiele einzeln zu betrachten in der heutigen Zeit meiner Meinung nach nicht mehr praktikabel ist. Man sollte stattdessen versuchen, sich spezielle Genres vorzunehmen und Metatheoreme für diese Genres zu entwickeln. Das Genre der Strategie- und Aufbauspiele scheint dabei besonders aussichtsreich zu sein. Zudem sollte man neben den Komplexitätsklassen NP und PSPACE noch weitere Klassen untersuchen. So existieren meines Wissens nach noch keine Metatheoreme für die Klassen EXPTIME und Co-NP.
Da wir in der vorliegenden Arbeit nur NP-Schwere und nicht NP-Vollständigkeit gezeigt haben, bleibt zu untersuchen, ob die jeweils angegebenen Spielelemente auch für die Schwere höherer Klassen ausreichend sind, oder, ob es Spiele in NP gibt, die diese Spielelemente enthalten.

Literaturverzeichnis

[ADGV15] Greg Aloupis, Erik D Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo Games are (Computationally) Hard. *Theoretical Computer Science*, 586:135–160, 2015.

[BDR98] Hendrik Blockeel and Luc De Raedt. Top-Down Induction of First-Order Logical Decision Trees. *Artificial intelligence*, 101(1):285–297, 1998.

[Cor04] Graham Cormode. The Hardness of the Lemmings Game, or Oh No, more NP-completeness Proofs. In *Proceedings of FUN*, volume 4, pages 65–76, 2004.

[CW03] Lourdes Pena Castillo and Stefan Wrobel. Learning Minesweeper with Multirelational Learning. In *IJCAI*, pages 533–540, 2003.

[DDO00] Erik D Demaine, Martin L Demaine, and Joseph O'Rourke. Push-Push and Push-1 are NP-hard in 2D. *arXiv preprint cs/0007021*, 2000.

[Dem01] Erik D Demaine. Playing Games with Algorithms: Algorithmic Combinatorial Game Theory. In *International Symposium on Mathematical Foundations of Computer Science*, pages 18–33. Springer, 2001.

[DH08] Erik D Demaine and Robert A Hearn. Constraint Logic: A Uniform Framework for Modeling Computation as Games. In *Computational Complexity, 2008. CCC'08. 23rd Annual IEEE Conference on*, pages 149–162. IEEE, 2008.

[DHH04] Erik D Demaine, Michael Hoffmann, and Markus Holzer. PushPush-k is PSPACE-complete. 2004.

[DHLN03] Erik D Demaine, Susan Hohenberger, and David Liben-Nowell. Tetris is hard, even to approximate. In *International Computing and Combinatorics Conference*, pages 351–363. Springer, 2003.

[Die96] Reinhard Diestel. *Graphentheorie*, volume 2. Springer Berlin, Heildelberg, New York, 1996.

[DVW16] Erik D Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. Is Harder/Easier than We Thought. 2016.

[For10] Michal Forišek. Computational Complexity of Two-Dimensional Platform Games. In *International Conference on Fun with Algorithms*, pages 214–227. Springer, 2010.

[GJ79] Michael R Gary and David S Johnson. Computers and Intractability: A Guide to the Theory of NP-completeness, 1979.

[Hop79] John E Hopcroft. *Introduction to Automata Theory, Languages and Computation: For VTU, 3/e*. Pearson Education India, 1979.

[Int07] xkcd: NP-complete. `http://xkcd.com/287/`, 2007. Aufgerufen am 13.11.2016.

[IPS82] Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton Paths in Grid Graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[Kay00] Richard Kaye. Minesweeper is NP-complete. *The Mathematical Intelligencer*, 22(2):9–15, 2000.

[KPS08] Graham Kendall, Andrew J Parkes, and Kristian Spoerer. A Survey of NP-complete Puzzles. *ICGA Journal*, 31(1):13–34, 2008.

[Mar85] Super Mario History. *Begleitheft der Super Mario All-Stars 25th Anniversary Edition*, page 7, 1985.

[Mar04] NFG Games + Nintendo Official Magazine Present - A Mario Sprite History. `http://nfggames.com/games/mariosprites/`, 2004. Aufgerufen am 13.11.2016.

[Mar14] Super Mario Bros. - Nintendo NES - Play Retro Games. `http://www.playretrogames.com/3171-super-mario-bros`, 2014. Aufgerufen am 13.11.2016.

[Mar15] NES - Super Mario Bros. - the Spriters Resource. `https://www.spriters-resource.com/nes/supermariobros/`, 2015. Aufgerufen am 13.11.2016.

[Mor96] Eduardo M Morales. Learning Playing Strategies in Chess. *Computational Intelligence*, 12(1):65–87, 1996.

[NES16] Sales Data - Hardware and Software Sales Units. `https://www.nintendo.co.jp/ir/en/sales/hard_soft/index.html`, 2016. Aufgerufen am 13.11.2016.

[NISI98] Tomofumi Nakano, Nobuhiro Inuzuka, Hirohisa Seki, and Hidenori Itoh. Inducing Shogi Heuristics using Inductive Logic Programming. In *International Conference on Inductive Logic Programming*, pages 155–164. Springer, 1998.

[P+79] J Plesn et al. The NP-completeness of the Hamiltonian Cycle Problem in Planar Diagraphs with Degree Bound Two. *Information Processing Letters*, 8(4):199–201, 1979.

[Pir16] Piranha Plant - Super Mario Wiki, the Mario Encyclopedia. `http://www.mariowiki.com/Piranha_Plant`, 2016. Aufgerufen am 13.11.2016.

[RFB00] Jan Ramon, Tom Francis, and Hendrik Blockeel. Learning a Go Heuristic with TILDE. In *International Conference on Computers and Games*, pages 151–169. Springer, 2000.

[Rob83] John Michael Robson. The Complexity of Go. In *IFIP Congress*, pages 413–417, 1983.

[Rob84] John Michael Robson. N by N checkers is Exptime complete. *SIAM Journal on Computing*, 13(2):252–267, 1984.

[Sam59] Arthur L Samuel. Some Studies in Machine Learning using the Game of Checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[Sch10] Das Rucksackproblem: schwache NP-Härte und Approximation. `http://algo.rwth-aachen.de/Lehre/WS0910/VBuK/Folien/komplexitaet6.pdf`, 2010. Aufgerufen am 26.11.2016.

[Sch14] Tobias Schmitz. Ebenfalls gelüftet: Wann Super Mario Bros. in Europa erschien. . *Nintendo-Online. 14. August 2014, abgerufen am 14. August 2014.*, 2014.

[SDS94] Nicol N Schraudolph, Peter Dayan, and Terrence J Sejnowski. Temporal Difference Learning of Position Evaluation in the Game of Go. *Advances in Neural Information Processing Systems*, pages 817–817, 1994.

[Shy16] Yoshi (Franchise) - Super Mario Wiki, the Mario Encyclopedia. `http://www.mariowiki.com/Yoshi_(franchise)`, 2016. Aufgerufen am 13.11.2016.

[SL06] István Szita and András Lörincz. Learning Tetris using the Noisy Cross-Entropy Method. *Neural computation*, 18(12):2936–2941, 2006.

[Tes89] Gerald Tesauro. Neurogammon wins Computer Olympiad. *Neural Computation*, 1(3):321–323, 1989.

[Tes95] Gerald Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[TSKY13] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N Yannakakis. The Mario AI Championship 2009-2012. *AI Magazine*, 34(3):89–92, 2013.

[Vig14] Giovanni Viglietta. Gaming Is a Hard Job, but Someone Has to Do It! *Theory of Computing Systems*, 54(4):595–621, 2014.

[Vig15] Giovanni Viglietta. Lemmings is PSPACE-complete. *Theoretical Computer Science*, 586:120–134, 2015.

[Yos15] Super Mario World 2: Yoshi's Island - Nintendo Super NES - Play Retro Games. `http://www.playretrogames.com/2380-super-mario-world-2-yoshi-s-island`, 2015. Aufgerufen am 13.11.2016.

[Yos16a] SNES - Super Mario World 2: Yoshi's Island - the Spriters Resource. `https://www.spriters-resource.com/snes/yoshiisland/`, 2016. Aufgerufen am 13.11.2016.

[Yos16b] Super Mario World 2: Yoshi's Island – Wikipedia. `https://de.wikipedia.org/wiki/Super_Mario_World_2:_Yoshi%E2%80%99s_Island`, 2016. Aufgerufen am 13.11.2016.

[Yos16c] Super Mario World 2: Yoshi's Island (Game) - Giant Bomb. `http://www.giantbomb.com/super-mario-world-2-yoshis-island/3030-2866/`, 2016. Aufgerufen am 27.11.2016.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift