Untersuchung der Lernfähigkeit verschiedener Verfahren am Beispiel von Computerspielen

Abschlussarbeit
zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)

Thilo Stegemann
s0539757
Angewandte Informatik

12. März 2017

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

Erstprüfer: Prof. Dr. Burkhard Messer
Zweitprüferin: Prof. Dr. Adrianna Alexander

Abstrakt

LOREM IPSUM

Abkürzungsverzeichnis

bzw. Beziehungsweise

eng. Englische Sprache

ID Identifikator

MDP Markov decision process

MEP Markov Entscheidungsprozess

UI User interface

vgl. Vergleich

vs. Versus, Gengenüberstellung

Abbildungsverzeichnis

Tabellenverzeichnis

Inhaltsverzeichnis

Kapitel 1

Projektvision

Geschichte 10 Sätze ...

1.1 Zielsetzung

Das Ziel der Arbeit ist es, ein bereits existierendes Lernverfahren zu implementieren und dessen Leistungsfähigkeit und Grenzen zu untersuchen. Das Lernverfahren soll eigenständig und automatisch eine Strategie lernen. Jeweils eine Strategie für das Strategiespiele Tic Tac Toe und das Strategiespiel Reversi. Wir bezeichnen die Implementierung des Lernverfahrens, als lernenden Agenten.

Ein weiteres Ziel in dieser Arbeit ist die Entwicklung von Bewertungsfunktionen (Heuristiken) für Reversi und Tic Tac Toe. Eine Heuristik berechnet eine Gewinnwahrscheinlichkeit ausgehend von einem Spielzustand. Ein Spielzustand mit einer hohen heuristischen Bewertung ist, gegenüber einem Spielzustand mit niedriger heuristischer Bewertung, zu bevorzugen. Eine Bewertungsfunktion soll das Spielwissen eines fortgeschrittenen menschlichen Spielers simulieren und als Implementierungsgrundlage für den nicht lernenden Agenten (auch heuristischer Agent) dienen.

1.2 Quantifizierung der Ziele

Die Leistungsfähigkeit und Grenzen, des Lernverfahrens, beurteilen wir anhand diverser Testspiele. Bei diesen Testspielen spielt der lernende Agent gegen den nicht lernenden Agenten und den Zufallsagenten. Die Agenten werden jeweils in den Strategiespielen Tic Tac Toe und Reversi gegeneinander antreten. Wir unterteilen die Testspiele in drei Phasen. In der ersten Phase (kurze Lernphase) lernt das Lernverfahren bzw. der lernende Agent in 100 Spielen gegen sich selbst eine Strategie. Wir erhören die Anzahl der Spiele gegen sich selbst in der zweiten Phase (mittlere

Lernphase) auf 1.000 Spiele und in der dritten Phase (lange Lernphase) auf 10.000 Spiele gegen sich selbst. Nach Abschluss jeder Phase muss der lernende Agent genau 100 Testspielen gegen den nicht lernenden Agenten absolvieren.

Wir testen die Leistungsfähigkeit der Bewertungsfunktionen ebenfalls anhand von Testspielen. Der nicht lernende Agent wird gegen einen Zufallsagenten antreten. Ein Zufallsagent wählt, aus allen möglichen Aktionen in einer Spielsituation, zufällig eine Aktion aus. Das Testkriterium der Bewertungsfunktionen ist eine Gewinnquote von mindestens 60% in 100 Testspielen gegen einen Zufallsagenten. Sollte der Agent mindestens 60% aller Testspiele Gewinnen, dann bezeichnen wir diesen, als Testgegner mit fortgeschrittenem Spielniveau.

1.3 Realisierung des Heuristik Agenten

In der Implementierung des nicht lernenden oder heuristischen Agenten ist, neben der Bewertungsfunktion, noch ein anderes Verfahren enthalten. Das Suchbaumverfahren für 2-Personenspiele. Dieses Verfahren durchsucht einen Spielbaum nach der bestmöglichen Aktion (einem Spielzug) in einem gegebenen Zustand. Ein Zustand oder Spielzustand ist eine Spielsituation bzw. eine Stellung der Spielfiguren auf dem Spielfeld.

Das Problem der Suchverfahren ist die Dimensionalität bzw. Komplexität des Ausgangsproblems. Suchbaumverfahren können für sehr einfache Probleme relativ schnell eine optimale Aktion finden. Die Größe des Suchbaums wächst exponentiell mit der Komplexität des Problems, d.h. die Laufzeit des Suchbaumverfahrens ohne Erweiterungen könnte für das Strategiespiele Tic Tac Toe nicht handhabbar sein und ist für das Strategiespiel Reversi nicht handhabbar. Wir schreiben "könnte" bei Tic Tac Toe, weil dieses noch ein recht einfacher Vertreter der Strategiespiele ist, dahingegen ist Reversi ein komplexeres Strategiespiel.

Um die Diemensionalitätsproblematik zu lösen, kombinieren wir Suchbaumverfahren mit Heuristiken, wir bezeichnen diese Kombination als heuristische Suche. Eine Heuristik berechnet eine Gewinnwahrscheinlichkeit, ausgehend von einem Spielzustand. Ein Spielzustand mit einer hohen heuristischen Bewertung ist, gegenüber einem Spielzustand mit niedriger heuristischer Bewertung, zu bevorzugen.

Das Suchbaumverfahren muss den Suchbaum, unter Verwendung einer Heuristik, nicht mehr komplett durchsuchen. Die Suche kann in einer bestimmten Suchbaumtiefe abgebrochen werden. Das Suchbaumverfahren liefert die erste Aktion

einer Aktionssequenz. Eine Aktionssequenz ist eine Folge von Aktionen und beschreibt einen Pfad im Suchbaum. Die Aktionssequenz, welche von der heuristischen Suche ausgewählt wurde, repräsentiert den Spielzustand mit der maximalen Gewinnwahrscheinlichkeit.

Die Qualität dieser Gewinnschätzung ist wiederum von der maximalen Suchtiefe und der Bewertungsfunktion abhängig. Eine größere Suchtiefe resultiert in einer besseren Schätzung, weil unter Umständen mehr Spielzustände berücksichtigt werden können. Die Verwendung einer Bewertungsfunktion ist keine Garantie für eine optimale Strategie. Verschiedene Bewertungsfunktionen können stark voneinander abweichende Gewinnschätzungen für Spielzustände berechnen.

1.4 Realisierung des TD-Q lernenden Agent

Wir stellen mehrere Lernverfahren innerhalb dieser Arbeit vor, aber wir werden nur das Q-Lernen (auch TD-Q-Lernen) implementieren und untersuchen. Das TD-Q-Lernen ist ein Lernverfahren aus dem Bereich des verstärkenden Lernens. Das TD-Q-Lernen soll es uns ermöglichen einen selbst lernenden Agenten zu programmieren. Verstärkendes Lernen (eng. reinforcement Learning) ist eine Lernkategorie des maschinellen Lernens. Problemstellungen des verstärkenden Lernens sind, u.a. das lernen von Strategiespielen, wie Schach, Reversi, Dame oder Backgammon. Der theoretische verstärkend lernende Lösungsansatz dieser Probleme ist wie folgt: ein Agent soll ein ihm unbekanntes Strategiespiel lernen (das Strategiespiel ist die unbekannte Umgebung), für einen Spielzug (Aktion) in einer Spielsituation (Zustand) erhält der Agent eine numerische Belohnung oder Bestrafung (Verstärkung), mittels dieser Verstärkung soll der Agent ein optimales Verhalten in der ihm unbekannte Umgebung erlernen.

Wie realisiert das TD-Q-Lernen diesen verstärkenden Lernansatz? Das TD-Q-Lernen lernt Q-Werte für Zustand/Aktionspaare, diese Q-Werte werden bei jedem erneuten Auftreten des Zustand/Aktionspaares aktualisiert. Eine Q-Funktion ist eine Abbildung von allen möglichen Zustand/Aktionspaaren auf Q-Werte und eine Q-Funktion ist eine Möglichkeit Nutzeninformationen zu speichern [RN12, S. 974]. Nachdem der Agent eine Q-Funktion gelernt hat, kann er mittels dieser, vermeintlich optimale Aktionen auswählen. Wir schreiben "Vermeintlich", weil eine gelernte Q-Funktion nicht immer zu einer optimalen Strategie konvergiert.

Wir zeige in dieser Arbeit praktisch, dass das TD-Q-Lernen ohne Erweiterungen, nur auf Probleme mit geringer Komplexität angewendet werden kann. Die Komplexität bzw. Dimensionalität des Ausgangsproblems ist ein Grund dafür, dass die

gelernte Q-Funktion nicht immer zu einer optimalen Strategie konvergiert, ein anderer Grund ist die zeitliche Beschränkung durch die Realität, d.h. in der Realität können nicht unendlich viele Testspiele durchgeführt werden. Es wurde bereits empirisch belegt, dass das Q-Lernen, sollte jedes Zustand/Aktionspaar nahezu unendlich oft besucht und aktualisiert werden, immer zu einer optimalen Strategie konvergiert. Das Problem dabei ist, dass die Komplexität bzw. die Dimensionalität des Ausgangsproblems, ein exponentiellen Verhältnis zur Zustands- und Aktionsmenge hat.

Lernt der TD-Q Agent, z.B. innerhalb von 10.000 Testspielen eine nahezu optimale Strategie für ein Tic Tac Toe Spiel mit 3 mal 3 Dimensionen (9 Spielfelder), dann ist das TD-Q-Lernen praktisch für ein Strategiespiel bis zu dieser Dimensionalität anwendbar. Erhöhen wir die Zustands- und Aktionsdimension, z.B. bei einem 16 Spielfelder Tic Tac Toe Spiel, dann reichen selbst 1.000.000 Testspiele unter Umständen nicht mehr aus, um eine annähernd optimale Strategie zu lernen. Jede weitere Dimension erhöht außerdem die Dauer eines Trainingsspiels, d.h. für jede weitere Dimension benötigt das TD-Q-Lernverfahren erheblich mehr Testspiele, um zu einer annähernd optimalen Strategie zu konvergieren und gleichzeitig erhöht sich die Dauert jedes Testspiels für jede zusätzliche Dimension des Ausgangsproblems.

1.5 Hypothese

Der Heuristik Agent wird in beiden Strategiespielen gegen den lernenden Agenten mindestens 50% aller Testspiele gewinnen.

Bestätigen wir diese Hypothese, dann belegen wir folgende Aspekte:

1. Das TD-Q-Lernen kann, innerhalb von maximal 10.000 Trainingsspielen gegen sich selbst, keine Strategie entwickeln, die in 100 Testspielen häufiger Gewinnt, als die in dieser Arbeit implementierte 2-Züge vorausschauende Heuristik-Suche.

2. Das TD-Q-Lernen muss möglicherweise mehr als 10.000 Trainingsspiele gegen sich selbst spielen, um eine bessere Strategie, als die nicht lernende Strategie, zu lernen.

3. Die Konvergenzgeschwindigkeit das TD-Q-Lernen zu einer optimalen Strategie, ist möglicherweise stark von der Dimensionalität des Ausgangsproblems abhängig, d.h. genau wie die uninformierten Suchbaumverfahren, ist das TD-Q-Lernen nur auf sehr einfache bzw. niedrig dimensionale Probleme anwendbar. Konvergenzgeschwindigkeit ist die Zeit, die ein Lernverfahren benötigt, bis es eine annähernd optimale Strategie entwickelt hat.

4. Das reine TD-Q-Lernen, ohne Erweiterungen, ist möglicherweise keine geeignetes Lernverfahren für das lernen eines Strategiespiels.

Kapitel 2

Strategiespiele und Spieltheorie

In diesem Kapitel werden die Strategiespiele Tic Tac Toe Abschnitt 2.1 und Reversi Abschnitt 2.2 vorgestellt und das jeweilige Regelwerk wird definiert. Weiterhin werden verschiedene Konzepte der Spieltheorie Abschnitt 2.3 genauer erklärt und veranschaulicht. Die 2.3.1 Minimax-Suche und die 2.3.2 Alpha-Beta-Kürzung sind Suchbaumverfahren die in Zweipersonenstrategiespielen eingesetzt werden. Die Alpha-Beta-Kürzung ist eine Verbesserung der Minimax-Suche und kein völlig anderes Suchverfahren. Es werden noch drei weitere Verbesserungsmöglichkeiten für die Minimax-Suche bzw. die Alpha-Beta-Suche erklärt. Die 2.3.3 Iterativ vertiefende Tiefensuche ist ein Verfahren, welches Breitensuche und Tiefensuche kombiniert und bis zu einer bestimmten Suchtiefe ein bestmögliches Ergebnis sucht. Die 2.3.4 Übergangstabellen beschreiben eine Möglichkeit Übergänge zu vermeiden. Übergänge sind identische Spielsituationen, die von Suchbaumverfahren als unterschiedliche Spielsituationen erkannt werden und daher redundante Suchbaumzweige darstellen. Das letzte, in dieser Arbeit behandelte, Konzept der Spieltheorie, ist die Heuristik. Eine 2.3.5 Heuristik ist eine Bewertungsfunktion B(s), welche für jeden Spielzustand (Stellung der Spielfiguren) s eine Schätzung bereitstellt. Die Bewertung sagt aus, wie Wertvoll ein Spielzustand s ist.

2.1 Das Strategiespiel Tic Tac Toe

Tic Tac Toe ist ein Spiel, welches von genau zwei Spielern gespielt wird. Während eines gesamten Spiels darf ein Spieler nur Kreuze setzen und der andere Spieler nur Kreise. Ein Spieler der während einer Partie nur Kreuze setzen darf wird als Kreuzspieler und sein Gegner als Kreisspieler bezeichnet. Wir können uns die Kreuze und Kreise als Spielsteine vorstellen, die sobald sie auf das Spielfeld gesetzt wurden, nicht mehr verändert oder verschoben werden können. Das Spielbrett ist eine 4 x 4 große Matrix, also können maximal 16 Spielsteine in diese Matrix gesetzt werden. Der Kreuzspieler muss immer als erster beginnen. Im ersten Spielzug stehen dem Kreuzspieler 16 mögliche Positionen zur Verfügung. Die Anzahl der möglichen Po-

sitionen reduziert sich jede Runde um 1, weil jede Runde genau ein gesetzter Spielstein ein Spielfeld besetzt. Folglich ist die maximale Länge einer Spielzugsequenz bei einem 4 x 4 Spielfeld gleich 16. Es ist auch möglich, dass das Spiel bereits vor der 16. Runde beendet wird.

Spielzüge jeder Spieler setzt abwechselnd entweder ein Kreuz oder einen Kreis in ein Spielfeld des Spielbretts. Ein Spielstein kann in jedes der 16 Spielfelder gesetzt werden, außer dieses ist bereits mit einem anderen Spielstein besetzt, dann muss der Spieler ein anderes Spielfeld auswählen. Die Spieler führen solange Ihre Spielzüge aus, bis eine Siegesformation eines Spielsteintyps erreicht ist oder alle Spielfelder besetzt sind.

Ziel des Spiels ist es vier Kreuze oder vier Kreise in einer bestimmten Position anzuordnen. Es existieren mehrere unterschiedliche Anordnungen von Spielsteinen, die das Spiel beenden und einen Sieg herbeiführen. Bei einem 4 x 4 Spielfeld existieren vier vertikale, vier horizontale und zwei diagonale Anordnungen der Spielfiguren, welche einen Sieg herbeiführen würden. Insgesamt zehn verschiedene Siegesanordnungen für beide Spieler. Sind alle Spielfelder besetzt und für keinen der Spieler ist eine Siegesformation aufgetreten, dann gewinnt beziehungsweise verliert keiner der beiden Spieler und es entsteht ein Unentschieden. Gewinnt ein Spieler mit einer Siegesanordnung seiner Spielsteine, dann verliert der andere Spieler dadurch automatisch in gleicher Höhe (Nullsummenspiel).

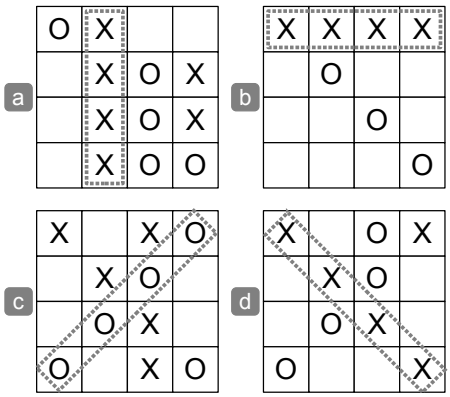

Abbildung 2.1 Veranschaulichtes Ziel des Strategiespiels Tic Tac Toe.

Vier mögliche Siegesformationen sind in Abbildung 2.1 dargestellt. (a) Der Kreuz-

spieler gewinnt knapp gegen seinen Kontrahenten mit einer ununterbrochenen vertikalen Anordnung seiner Spielsteine. Der Kreisspieler hätte fast eine diagonale Reihe aus Kreisen verbunden, diese wurde jedoch vom Kreuzspieler mit einem Spielstein unterbrochen. Zudem hätte der Kreisspieler auch fast eine vertikale Reihe ohne Unterbrechungen vervollständigt, aber der Sieg des Kreuzspielers hat die Partie vorher beendet. (b) Der Kreuzspieler erreicht eine horizontale Siegesanordnung, vier Kreuzsteine in einer horizontalen Zeile angeordnet. (c) Der Kreisspieler besiegt den Kreuzspieler mit einer diagonalen Siegesanordnung. (d) Der Kreuzspieler gewinnt ebenfalls durch eine diagonale Siegesformation seiner Spielsteine.

2.2 Das Strategiespiel Reversi

Das Spiel Reversi oder auch Othello genant, wird auf einem 8 x 8 Spielbrett gespielt. Es ist ein Spiel für zwei Personen die gegeneinander antreten. Eine Person setzt weiße runde Spielsteine und die andere Person schwarze runde Spielsteine. Jede neue Partie Reversie beginnt im selben Ausgangszustand (siehe Abbildung 2.2). Die Spieler setzen nacheinander in ihren Spielzügen genau einen Spielstein. Wie beim klassischen Tic Tac Toe aus Abschnitt **??** behalten die Spieler während des gesamten Spiels ihre Spielsteinfarbe und einmal gesetzte Spielsteine können ihre Position nicht mehr verändern.

Anmerkung zu Abbildung 2.2 Die äußeren weiß hinterlegten Reihen in denen sich Zahlen befinden, dienen dazu die Positionen der einzelnen Spielfelder genau zu definieren. In der Ausgangsspielsituation befinden sich bereits 2 weiße Spielsteine an den Positionen (3,4) und (4,3) und zwei schwarze Spielsteine an den Positionen (3,3) und (4,4).

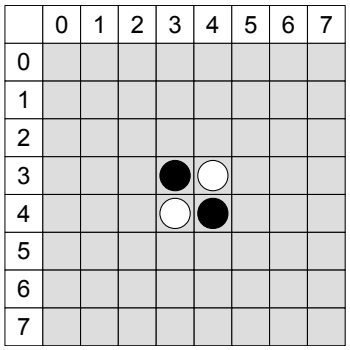

Abbildung 2.2 Ausgangsspielzustand Reversi.

Eine Besonderheit von Reversi ist, dass gesetzte Spielsteine ihre Farbe ändern

können. Werden z.B. zwei weiße Spielsteine von zwei schwarzen in einer horizontalen Linie eingeschlossen, dann werden die weißen Spielsteine in schwarze umgewandelt beziehungsweise umgedreht. Das erobern der gegnerischen Spielsteine ist vom aktuell gesetzten Spielstein abhängig.

Spielzüge sind bei Reversi nicht beliebig, sie unterliegen bestimmten Regellungen. Eine Regel für das Setzen eines Spielsteins ist, nur wenn mindestens ein gegnerischer Spielstein erobert wird, darf ein Spielstein an diese Stelle gesetzt werden. Weiterhin darf ein Spielstein nur dann gesetzt werden wenn, ein anderer Spielstein (Anker), mit der gleichen Farbe, in einer diagonalen, vertikalen oder horizontalen Linie, existiert. Es dürfen auch keine freien Felder zwischen dem zu setzendem Stein und dem Anker liegen. Ein Anker ist ein Spielstein mit der selben Farbe wie der zu setzende Spielstein. Ein zu setzender Spielstein kann mehrere Anker haben, aber er muss mindestens einen und kann maximal acht Anker haben.

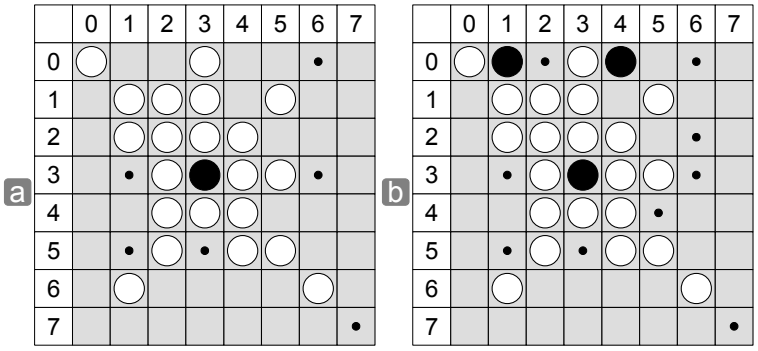

Abbildung 2.3 Spielzugmöglichkeiten Reversi.

Anmerkung zu Abbildung 2.3, diese zeigt zwei möglicherweise nicht in der Praxis auftretende Spielsituationen, die einzig verdeutlichen sollen welche Zugmöglichkeiten der Spieler mit den schwarzen Spielsteinen hat und warum nur diese Züge möglich sind. Die kleinen schwarzen Punkte zeigen die Positionen an denen ein schwarzer Spielstein gesetzt werden darf. (a) Eine Spielsituation mit maximal einem möglichen Anker. (b) Eine Spielsituation mit maximal 3 möglichen Ankern für die Position (3,1).

Der Anker in Abbildung 2.3 (a) ist der schwarze Spielstein an der Stelle (3,3). In diesem Beispiel soll schwarz am Zug sein und einen Spielstein platzieren. Die Positionen (3,1) und (3,6) ermöglichen eine horizontale, (5,3) ermöglicht eine vertikale und (5,1), (0,6) und (7,7) ermöglichen eine diagonale Verbindung mit dem Anker auf Position (3,3). Die meisten gegnerischen Spielsteine könnte schwarz erobern, indem er seinen Spielstein auf das Spielfeld (7,7) setzt. In Abbildung 2.3 (b) setzt der

Spieler seinen schwarzen Spielstein an die Position (3,1), dann hat dieser 3 Anker. Einen vertikalen Anker (0,1), einen horizontalen Anker (3,3) und einen diagonalen Anker (0,4). Insgesamt würden 5 weiße Spielsteine erobert werden, also 2 mehr als in (a) maximal möglich wären.

Ziel des Spiels ist es, am Ende des Spiels mehr Spielsteine seiner eigenen Farbe zu haben, als der Gegner Spielsteine in seiner Farbe hat. Das Spiel endet, wenn keiner der beiden Spieler mehr einen Spielstein, nach den Regeln des Spiels, auf das Spielbrett setzen kann.

2.3 Spieltheorie

Schach, Vier Gewinnt, Dame, Tic Tac Toe und Reversi sind strategische Spiele für zwei Personen, die gegeneinander antreten, um nach den Regeln des Spiels, den Gegenspieler zu besiegen. Diese Spiele sind deterministisch weil, dass Spiel nicht vom Zufall abhängt und der gleiche Spielzug führt bei gleichem Ausgangszustand immer zum selben Zustandsübergang. Zudem sind sie überschaubar weil, zu jedem Zeitpunkt des Spiels das Spielfeld und alle Spielzüge einsehbar sind. Ein nichtdeterministisches Spiel mit Gegenspieler ist z.B. Backgammon, denn Würfelergebnisse und somit der Zufall sind Bestandteil des Spiels. Wie kann ein Programm einen Menschen in einem dieser Strategiespiele besiegen? In den nachfolgenden Unterabschnitten werden Konzepte der Spieltheorie erläutert, die versuchen diese Frage zu beantworten.

Nullsummenspiele

Ein Nullsummenspiel ist ein Spiel bei dem der Verlust eines Spielers einen gleich hohen Gewinn für den Gegenspieler bedeutet. Gewinnt ein Spieler eine Partie eines Nullsummenspiels, dann verliert der Gegenspieler automatisch. Der gewinnende Spieler erhält einen Pluspunkt und der verlierende Spieler einen Minuspunkt. Die Summe der beiden Ergebnisse ist Null. Gewinnt beziehungsweise verliert keiner der beiden Spieler, dann erhalten beide Spieler als Spielergebnis eine Null. Die Summe der beiden Ergebnisse ist wiederum Null, so lässt sich der Name Nullsummenspiele"herleiten. Die beiden Spiele Tic Tac Toe und Reversi sind Nullsummenspiele.

2.3.1 Minimax

Ein Spieler wird als MAX bezeichnet und der Gegenspieler als MIN. Spieler MAX versucht einen maximalen Gewinn für sich zu erlangen und Spieler MIN versucht den erreichbaren Gewinn von MAX zu minimieren. Es wäre auch möglich anzunehmen, dass Spieler MIN einfach irgendeinen zufälligen oder dummen Spielzug auswählt und MAX einen einfachen Sieg erlangt. Diese Annahme entspricht jedoch wenig einem realen Spiel, bei dem beide Spieler versuchen das Spiel zu gewinnen.

In Abbildung 2.6 ist der Ablauf einer Minimax-Suche veranschaulicht. Der Minimax-Suchbaum berücksichtigt jeden Zustand indem sich die Spielwelt befinden kann. Im ersten Spielzug könnte Spieler MAX sein Kreuzspielstein in die obere linke Ecke setzen, daraus ergeben sich neue Zustandsmöglichkeiten. Spieler MIN könnte seinen Kreisspielstein ein Feld weiter rechts und in die selbe Reihe wie Spieler MAX setzen. Der Nutzen der einzelnen Züge ist zur Zeit der Ausführung noch nicht bekannt, erste wenn das Tic Tac Toe Spiel einen Endzustand erreicht, werden den Spielern ihre Spielergebnisse mitgeteilt. Der Minmax-Suchbaum ist somit rekursiv zu betrachten. Von seinen Blattknoten ausgehend entscheidet sich MIN für den geringsten Nutzwert und MAX für den höchsten. Die Entscheidungen stehen in direkter Abhängigkeit zur vorherigen Entscheidung des Gegenspielers.

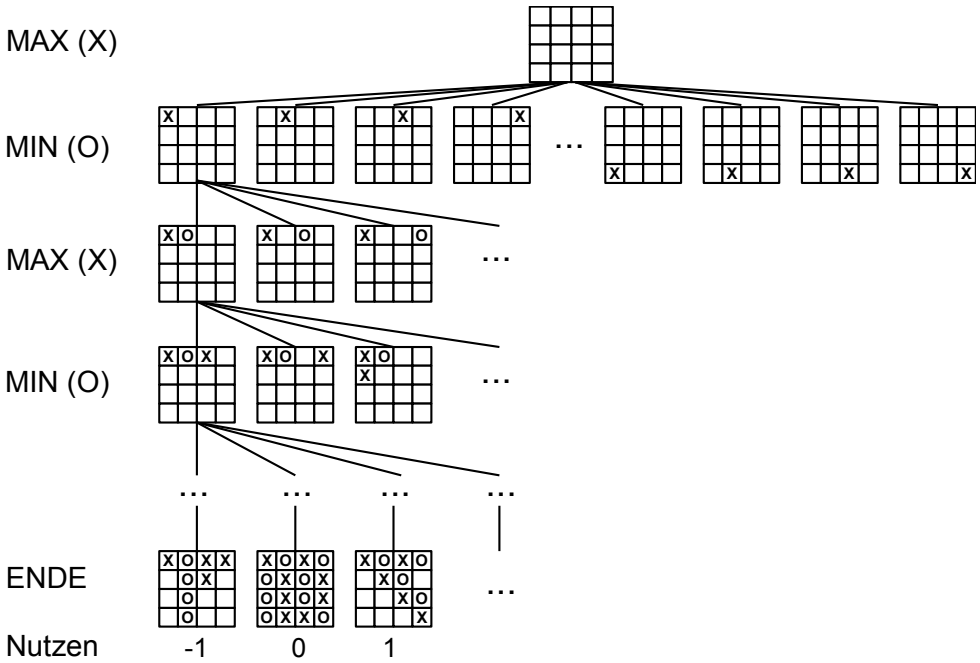

Abbildung 2.4 Ein (partieller) Suchbaum vgl. [RN12, S. 208]

Das größte Problem der Minimax-Suche ist die exponentielle Vergrößerung der Anzahl der Blattknoten des Suchbaums, schon bei sehr einfachen Spielen z.B. einem 3x3 Tic Tac Toe mit Neun Spielfeldern ist der Suchbaum bereits sehr groß ungefähr 362880 Blattknoten mit einem effektiven Verzweigungsfaktor von Neun der sich nach jedem Halbzug um Eins verringert. Erweitern wir die Dimension des Tic Tac Toe Spiels, so erhalten wir ein 4x4 Tic Tac Toe Spiel mit 16 Spielfeldern, einem Verzweigungsfaktor von 16 und ungefähr 20922789888000 Blattknoten. Der effektive Verzweigungsfaktor beim Schach liegt etwa bei 30 bis 35. Bei einem typischen Spiel mit 50 Zügen pro Spieler hat der Suchbaum dann mehr als $30^{100} \approx 10^{148}$ Blattknoten[Ert16, S. 114]. Der Minimax-Algorithmus ist, auf Grund enormer Rechenzeit, in seiner Reinform praktisch nicht anwendbar, daher werden wir Erweiterungen der Minmax-Suche und andere Konzepte kennen lernen.

2.3.2 Alpha-Beta-Kürzung

Eine Möglichkeit die Rechenzeit der Minimax-Suche zu verbessern ist das Kürzen oder Beschneiden des Suchbaums(eng. Pruning). Beim Alpha-Beta-Kürzen wird der Teil des Suchbaums beschnitten, der keinen Effekt auf das Ergebnis der Minimax Suche hat. Der Minimax Algorithmus wird um zwei Parameter Alpha und Beta ergänzt. Die Bewertung erfolgt an jedem Blattknoten des Suchbaums. Alpha enthält den aktuell größten Wert, für jeden Maximum Knoten, der bisher bei der Traversierung des Suchbaums gefunden wurde. In Beta wird für jeden Minimum Knoten der bisher kleinste gefundene Wert gespeichert. Ist Beta an einem Minimum Knoten kleiner oder gleich Alpha ($Beta \leq Alpha$), so kann die Suche unterhalb von diesem Minimum Knoten abgebrochen werden. Ist Alpha an einem Maximum Knoten größer oder gleich Beta ($Alpha \geq Beta$), so kann die Suche unterhalb von diesem Maximum Knoten abgebrochen werden [Ert16, S. 116].

Verdeutlichen wir das Alpha-Beta-Pruning an Hand eines Beispiels aus dem Standardwerk der künstlichen Intelligenz von S. Russell und P. Norvig Abbildung 5.5 [RN12, S. 213]. Ein Dreieck mit der Spitze nach oben ist ein Maximumknoten und ein Dreieck mit der Spitze nach unten ist ein Minimumknoten. Leere Dreiecke ohne einen bezeichnenden Buchstaben und gestrichelter Umrandung sind noch nicht explorierte Knoten. Durchgängige Linien verweisen auf bereits besuchte Pfade und gestrichelte Linien verweisen auf noch nicht besuchte Pfade. Die Zahlen unterhalb der Blattknoten sind die Nutzwerte die der maximierende Spieler erhält, wenn er den Pfad bis zu diesem Blattknoten durchschreitet.

(a) Minimum Knoten B findet einen Nutzwert 3, da dieser Wert der bisher kleinste gefundene Wert ist wird er in Beta gespeichert.

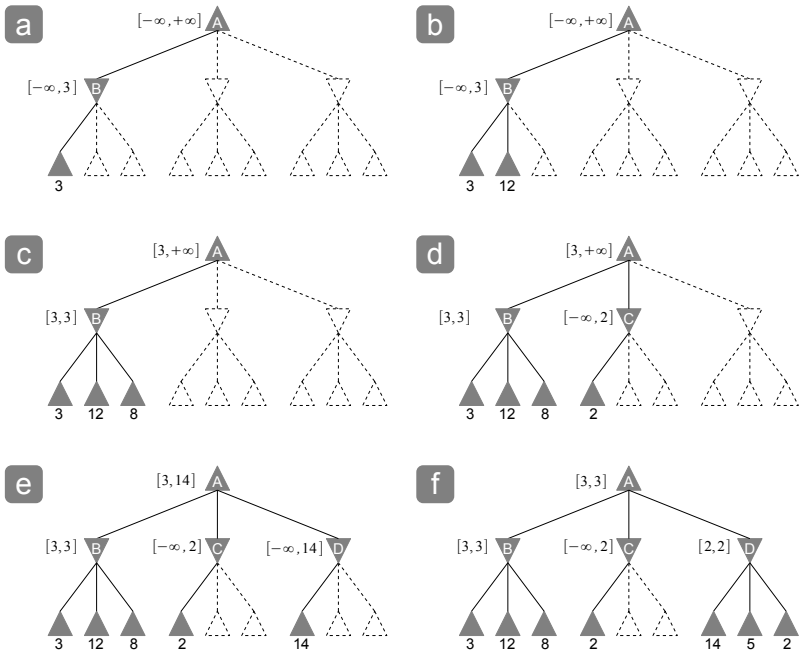

Abbildung 2.5 Ein Alpha Beta Suchbaum [RN12, S. 213].

(b) Der Minimum Knoten B exploriert einen zweiten möglichen Nutzwert 12. Dieser Wert ist höher als der vorher gefundene und in Beta gespeicherte Wert 3, daher wird der minimierende Spieler versuchen diesen Nutzwert für den maximierenden Spieler zu vermeiden. Der neue Wert wird vom Minimum Knoten B ignoriert und Beta bleibt unverändert.

(c) Minimum Knoten B findet den Wert 8, dieser ist genau wie 12 größer als 3 und daher wird Spieler MIN vermeiden, dass Spieler MAX zu diesem Spielergebnis gelangt. Minimum Knoten B hat alle seine nachfolgenden Knoten exploriert. Maximum Knoten A wird vom Minimum Knoten B maximal den Nutzwert 3 erhalten, somit ergibt sich für den Maximum Knoten A, dass dieser mindestens den Nutzwert 3 erreichen kann.

(d) Ein weiterer Minimum Knoten ist C. Der erste Blattknoten von C liefert einen Nutzwert von 2, weil dieser Wert der erste gefundene Wert unterhalb des Minimum Knotens C ist, wird er in Beta gespeichert. C wird Maximum Knoten A maximal einen Nutzwert 2 liefern. A wiederum kann durch Minimum Knoten B bereits einen minimalen Nutzwert von 3 erhalten und hat diesen in Alpha gespeichert. Es gilt $Beta \leq Alpha$ und es ist nicht notwendig die Knoten unterhalb von C weiter zu explorieren. Selbst wenn ein größerer Nutzwert gefunden werden würde, entscheidet sich der minimierende Spieler trotzdem für den kleineren Wert und würde ein

kleinerer Nutzwert als 2 gefunden werden, dann entscheidet sich der maximierende Spieler für den Nutzwert 3, den Minimum Knoten B liefert. Folglich kann der Suchbaum an dieser Stelle abgeschnitten werden, weil weitere gefundene Nutzwerte keinen Einfluss mehr auf das Ergebnis haben.

(e) Der letzte von A zu erreichende Minimum Knoten wird exploriert. Der erste Blattknoten unterhalb des Minimum Knoten D liefert den Nutzwert 14. Dieser Wert wäre für Maximum Knoten A eine starke Verbesserung, weil dieser bisher nur maximal einen Nutzwert von 3 erreichen konnte. Der minimierende Spieler hat noch zwei weitere Möglichkeiten(Knoten) zu explorieren und daher wird er versuchen einen geringeren Nutzwert als 14 zu finden.

(f) Minimum Knoten D findet in den beiden letzten Blattknoten die Nutzwerte 5 und 2. Der minimierende Spieler wählt die Möglichkeit mit dem geringsten Nutzwert 2. Dieser Nutzwert wird zum neuen Beta Wert. Der Suchbaum wird unterhalb vom Minimum Knoten D jedoch nicht abgeschnitten, weil der Nutzwert 2 erst im zuletzt explorierten Knoten gefunden wurde. Theoretisch könnten zwei Pfade unterhalb des Minimum Knoten D abgeschnitten werden, wenn der Blattknoten mit dem Nutzwert 2 zuerst exploriert worden wäre.

2.3.3 Iterativ vertiefende Tiefensuche

Die iterativ vertiefende Suche (eng. Iterative Deepening) ist eine Kombination der Breitensuche und der Tiefensuche. Diese Suchverfahren sind uninformierte (blinde) Suchverfahren. Die Strategien der uninformierten Suchverfahren haben keine zusätzlichen Informationen über Zustände, außer den in der Problemdefinition vorgegebenen. Alles was sie tun können, ist, Nachfolger zu erzeugen und einen Zielzustand von einem Nichtzielzustand zu unterscheiden. Die Reihenfolge der Suche ist entscheidend für die Unterscheidung der einzelnen uninformierten Suchverfahren [RN12, S. 116].

Die Breitensuche expandiert (erweitert oder vergrößert) zu erst alle Nachfolger (Knoten eines Suchbaums) die in derselben Tiefe liegen, beginnend mit dem Wurzelknoten. Sind alle Nachfolger einer Tiefe expandiert, dann werden deren Nachfolger nacheinander expandiert. Diesen Schritt wiederholt die Breitensuche bis ein gesuchtes Ergebnis gefunden wird.

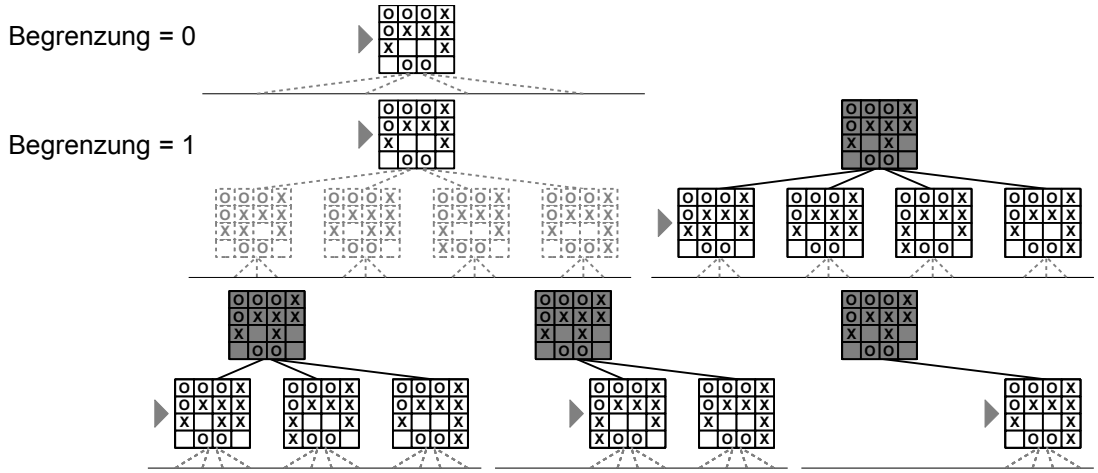

Abbildung 2.6 TODO

Die Tiefensuche expandiert zuerst die tiefsten Knoten des Suchbaums(Depth-first). Erreicht die Tiefensuche einen Endknoten der nicht dem gesuchten Ergebnis entspricht, dann werden die alternativen Knoten des letzten expandierten Knotens, der sich eine Tiefenebene höher befindet, expandiert.

Kombinieren wir diese beiden uninformierten Suchverfahren miteinander und mit einer Grenze für die Suchtiefe, erhalten wir die iterative Tiefensuche. Diese expandiert zuerst die Nachfolger des Wurzelknotens der Suchtiefe 1. Sind alle Knoten auf dieser Ebene expandiert, dann wird die Schranke für die aktuelle Suchtiefe um 1 erhöht (Iteration) und die Knoten der Suchtiefe 2 werden expandiert. Diese Schritte wiederholt die Tiefensuche bis ein Ziel gefunden wird.

Anwendung findet die iterative Tiefensuche bei der Zugsortierung für die Verbesserung des Alpha-Beta Suchbaumkürzens.

2.3.4 Übergangstabellen

Eine Übergangstabelle (eng. transition table) ist eine Tabelle in der Spielsituationen mit verschiedenen Attributen gespeichert werden (vgl. [RN12, S. 215 f.]). Übergänge sind der Grund dafür, dass der gleiche Spielzustand durch unterschiedliche Spielzugsequenzen auftritt (siehe Abbildung 2.7). Folgende Attribute werden in der Übergangstabelle gespeichert: der Spielzustand, die Alpha-Beta Werte, der bestmögliche Halbzug, der Nutzen und welcher Spieler gerade setzen muss. Alle Attribute beziehen sich auf den gespeicherten Spielzustand z.B. der bestmögliche berechnete Halbzug der für den gespeicherten Spielzustand möglich ist.

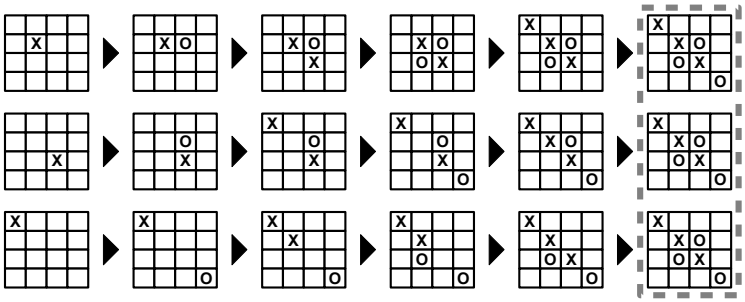

Abbildung 2.7 Verschiedene Spielzugsequenzen enden im selben Spielzustand.

Übergänge innerhalb des Suchbaums verursachen Redundanzen. Für jede dieser Redundanzen wird eine erneute Suche durchgeführt, falls diese nicht durch Alpha-Beta-Kürzung abgeschnitten werden. Sollten diese Übergänge vermieden werden können, dann würde sich die Rechenzeit der Suchverfahren weiter verringern, weil weniger Spielzustände durchsucht bzw. expandiert werden müssen. Wie können wir Spielzustände in einer Übergangstabelle abspeichern?

Zobrist Hash

Wenn ein Computerprogramm einen Gegenstand in einer großen Tabelle speichert, muss die Tabelle zwangsläufig durchsucht werden, um den Gegenstand wiederzuverwenden bzw. zu referenzieren. Dies gilt solange, bis eine Tabellendresse aus dem Gegenstand selbst, in systematischer weise, berechnet werden kann. Eine Funktion die Gegenstände in Adressen umwandelt ist ein Hash-Algorithmus, und die daraus resultierende Tabelle ist eine Hashtabelle [Zob70, S. 3].

In Abbildung 2.8 wird das Zobrist Hash Verfahrens auf den redundanten Spielzustand aus Abbildung 2.7 angewendet. (2.8 a) Wir weisen jedem Spielfeld zwei zufällige ganzzahlige Werte zu im Bereich von 0 bis maximal 1×10^9. Einen zufälligen Wert für den Kreuzspielstein an dieser Position und einen für den Kreisspielstein. Das 4x4 Tic Tac Toe Spielbrett sollte insgesamt 32 verschiedene Werte erhalten. (2.8 b) Dieser Spielzustand soll in einen Zobrist-Hash umgewandelt werden.

(2.8 c) Der Zobrist-Hash berechnet sich wie folgt, ist die aktuelle Position mit einem Kreuzspielstein oder einem Kreisspielstein besetzt, dann wähle den entsprechenden Wert aus (2.8 a). Dies wiederhole für jedes besetzte Spielfeld. Auf den bereits ermittelten Wert und den hinzukommenden Wert wird ein exklusives bitweises Oder (XOR) angewendet. In Python ist das Zeichen für die Funktion des exklusiven bitweisen Oder ein Zirkumflex (siehe Operatoren in der Berechnung 2.8 c). Das Ergebnis ist eine Adresse die exakt den Spielzustand (2.8 b) referenziert.

a

X = 660640090 O = 601151343	X = 651080001 O = 550176261	X = 707754336 O = 30179116	X = 240651458 O = 515695098
X = 843817469 O = 625774421	X = 446956442 O = 409234428	X = 888791315 O = 906370688	X = 10057952 O = 962066669
X = 925070678 O = 747101521	X = 179513842 O = 89793577	X = 538866973 O = 222479865	X = 144262103 O = 353844301
X = 595995309 O = 751411292	X = 883501364 O = 531273511	X = 727572818 O = 91717317	X = 7191668 O = 704554166

b

X			
	X	O	
	O	X	
			O

c

660640090 ^ 446956442 ^ 906370688 ^ 89793577 ^ 538866973 ^ 704554166 = **125309938**

Abbildung 2.8 Zobrist Hashing von Spielzuständen.

Spielzustände die gerade expandiert werden, können in die Übergangstabelle eingetragen werden, sollten diese nicht bereits in der Tabelle vorhanden sein. Ist dieser Spielzustand bereits in der Tabelle vorhanden, dann können die vorgeschlagenen besten Halbzüge aus der Tabelle ausgelesen und angewendet werden.

2.3.5 Heuristik

Eine Heuristik oder Bewertungsfunktion berechnet einen Nutzwert für einen gegebenen Spielzustand. Dieser Nutzwert gibt an, wie "wertvoll"diese Spielsituation hinsichtlich eines Sieges ist, sprich sie gibt an ob der Spieler in diesem Spielzustand eher gewinnen oder verlieren könnte. Trotz aller Optimierungen des Minimax Verfahrens (Alpha-Beta, Zugsortierung, Vermeidung von Redundanzen) wäre die Rechenzeit, für den zu durchsuchenden Baum, immer noch enorm hoch. Reale Zeitbeschränkungen z.B. bei Schach Spielen erlauben ein überaus langes Berechnen ohnehin nicht. Die Lösung ist das verwenden einer Heuristik. Der Kompromiss bei einer Heuristik ist: das Ergebnis wird geschätzt und ist nicht mehr sicher, aber die Suche kann nach einem Zeitkriterium abgebrochen werden und die beste bisher gefundene Lösung wird zurückgegeben. Sollte die Bewertungsfunktion für einen Spielzustand z.B. einen sehr hohen Wert berechnen, dann besagt dieser, der Spieler der diesen Spielzustand erreicht wird wahrscheinlich gewinnen.

Die Qualität einer Heuristik ist ausschlaggebend für die Spielerischen Fähigkeiten eines Programms. Ein Programm welches, durch eine schlechte Stellungsbe-

wertung (Heuristik) einen fatalen Spielzug des Gegners übersieht oder ignoriert, würde gegen ein Programm verlieren, welches diese Stellungen (Spielzustände) erkennt und ausnutzt bzw. entsprechend verhindert. Eine Bewertungsfunktion B(s) für ein Schachspiel enthält folgende Elemente, wobei s der Parameter für den Spielzustand ist[Ert16, S. 119]:

B(s) = a_1 x Material + a_2 x Bauernstruktur + a_3 x Königssicherheit
+ a_4 x Springer im Zentrum + a_5 x Läufer Diagonalabdeckung + ...,

das mit Abstand wichtigste Feature (Merkmal) Materialnach der Formel

Material = Material(eigenes Team) - Material(Gegner)

Material(Team) = Anzahl Bauern(Team) x 100 + Anzahl Springer(Team) x 300
+ Anzahl Läufer(Team) x 300 + Anzahl Türme(Team) x 500
+ Anzahl Damen(Team) x 900

Diese Schach Heuristik ist entstanden aus der Zusammenarbeit von Schachexperten und Wissensingenieuren. Die Schachexperten verfügen über Wissen und Erfahrungen bezüglich des Schachspiels, der Strategien, guter Zugstellungen und schlechter Zugstellungen. Der Wissensingenieur hat die meist sehr schwierige Aufgabe dieses Wissen in eine, für ein Programm, anwendbare Form zu bringen (vgl. [Ert16, S. 118]).

Eine Heuristik ist stark Abhängig von ihrer Grundlage, d.h. eine Heuristik die für ein Schachspiel konzipiert wurde, berücksichtigt die Spielfiguren, das Spielfeld und die Spielregeln des Schachspiels. Wir können diese Heuristik nicht direkt auf Reversi oder Tic Tac Toe anwenden, weil eine Schach Heuristik nicht auf andere Spiele angewendet werden kann. Eine Reversi Heuristik kann z.B. auch nicht für das Spielen eines Tic Tac Toe Spiels verwendet werden. Es ist uns jedoch möglich das Konzept, sprich die Essenz der Heuristik Entwicklung, aus dieser Beispielheuristik für Schachspiele zu entnehmen und auf die Heuristik Entwicklung für Reversi und Tic Tac Toe anzuwenden.

Mehr zum Thema Heuristik Entwicklung in Kapitel 5 Modellierung und Entwurf, in diesem Kapitel werden wir ähnlich der gerade vorgestellten Heuristik, eigene Bewertungsfunktionen für Tic Tac Toe und Reversi entwerfen und in Kapitel **??** Implementierung, werden die modellierten Bewertungsfunktionen praktisch angewendet.

Tic Tac Toe Heuristik

Reversi Heuristik

Kapitel 3

Einführung in verstärkendes Lernen

In diesem Kapitel behandeln wir Konzepte und Verfahren des maschinellen Lernens. Im letzten Abschnitt "Verstärkendes Lernen"behandeln wir das Konzept eines Agenten der in eine ihm unbekannte Umgebung ausgesetzt wird und in dieser ein optimales Verhalten lernen soll.

3.1 Verstärkendes Lernen eine Definition

Verstärkendes oder auch bestärkendes Lernen (eng. reinforcement Learing) beschäftigt sich mit dem Problem, dass ein Agent, innerhalb einer ihm unbekannten Umgebung, Aktionen (Entscheidungen) ausführen muss (Abbildung 3.1). Das Ziel des Agenten ist es, einen nummerischen Wert zu maximieren. Dieser Wert wird durch Belohnung oder Bestrafung (Verstärkung) verändert, dadurch lernt der Agent die Zusammenhänge zwischen den möglichen Aktionen in einem Zustand und deren Verstärkungen.

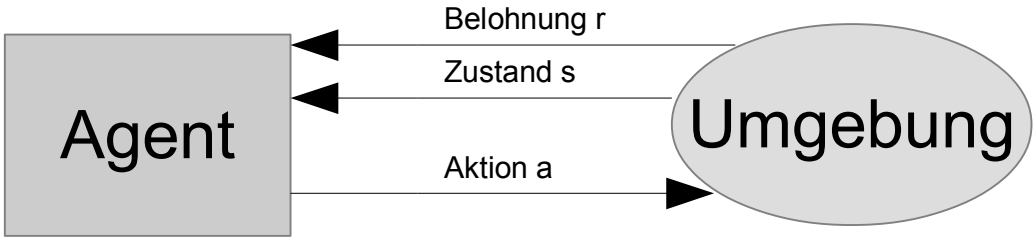

Abbildung 3.1 Der Agent und seine Wechselwirkung mit der Umgebung vgl. [Ert16, S. 290] und [Alp08, S. 398].

Wie kann sich der Agent die Bewertungen seiner Aktionen merken und sich daran erinnern? Dies ist möglich, weil der Agent über einen Erfahrungsspeicher verfügt z.B. eine Übergangstabelle. Der Agent schreibt neue Zustände, ausgeführte

Aktionen und mögliche Bewertungen der Aktionen in die Tabelle. Vorher prüft der Agent ob der Zustand bereits in der Tabelle eingetragen ist. Existiert dieser Zustand in der Tabelle, hat der Agent Erfahrung in diesem Zustand gesammelt und kann diese nutzen und anpassen.

Bei realen Problemen, wie dem lernen von Schach oder Reversi, erfolgt eine Belohnung oder Bestrafung nicht direkt nach einer Aktion des Agenten (verspätete Belohnung, eng. delayed reward). Erst nach Abschluss einer Partie, also nach einer Sequenz von bestimmten Aktionen, endet das Spiel und der Agent wird für einen Sieg belohnt oder für eine Niederlage bestraft. Eine große Schwierigkeit ist diese verspätete Belohnung am Ende einer Aktionssequenz auf die einzelnen Aktionen des Agenten aufzuteilen. Dieses Problem ist bekannt als Anerkennungszuweisung Problem (eng. credit assignment problem).

3.2 Fallbeispiel: Ein Agent im Labyrinth

Nehmen wir an es existiere folgender Agent, er kann vier Aktionen ausführen, bewege dich nach oben, unten, rechts oder links und er wird in einem ihm unbekannten Labyrinth ausgesetzt. Das Labyrinth ist die Umgebung und die Zustände der Umgebung verändern sich durch die Aktionen des Agenten (Übergänge), das heißt verändert der Agent seine Position innerhalb des Labyrinths, dann wechselt er von einem Ausgangszustand s, durch eine Aktion a, in einen neuen Zustand s'. Eine Aktion a ist immer Element der Menge aller möglichen Aktionen A. Welche Aktionen in einem bestimmten Zustand s möglich sind, wird durch die Funktion A(s) oder ACTIONS(s) bestimmt. Die Menge aller möglichen Zustände einer Umgebung bezeichnen wir als S, d.h. jeder einzelne Zustand s ist Element von S.

Der Agent lernt also, dass die Aktion 'bewege dich nach oben' den Ausgangszustand s in einen neuen Zustand s' transformiert. In einer deterministischen Umgebung wird der neue Zustand s' durch die Übergangsfunktion $\delta(s, a)$ bestimmt, sprich führt der Agent eine Aktion a in einem Zustand s aus, dann wird er definitiv den Zustand s' erreichen. Ist die Umgebung nicht deterministisch, dann verändert sich die Übergangsfunktion in P(s' | s,a), d.h. die Umgebung bestimmt die Wahrscheinlichkeit P mit der s' erreicht werden kann, wenn im Zustand s die Aktion a ausgeführt wird. Die Funktion P kann auch ein deterministisches Modell der Welt darstellen, wenn die Wahrscheinlichkeit jedes Zustandsübergangs 100% ist, also wenn nur ein Zustandsübergang pro Zustand/Aktions-Paar möglich ist.

Führt der Agent die Aktion 'bewege dich nach oben' aus, dann ist jedoch die

Zustandsveränderung abhängig von der individuellen Umgebung in der sich der Agent befindet, d.h. die Übergangsfunktion δ oder P(s' | s,a), wird von der Umgebung festgelegt und nicht vom Agenten. Diese Übergangsfunktionen werden auch als Modelle der Umgebung bzw. der Welt bezeichnet. In einem Labyrinth kann der Agent nicht immer alle seiner vier Aktionen ausführen, denn er ist umringt von Mauern die seinen Aktionsradius beschränken. Würde er trotzdem eine dieser Aktion ausführen, dann verändert sich der Zustand der Umgebung nicht, denn der Agent würde sprichwörtlich "gegen die Wand laufen".

Nach einer endlichen Sequenz von Aktionen (Zustandsfolge oder Umgebungsverlauf) gelingt es dem Agenten den Ausgang des Labyrinths zu erreichen und er erhält eine nummerische Belohnung (eng. reward), die Belohnung kann auch als Verstärkung (eng. reinforcement) oder Gewinn bezeichnet werden. Eine Gewinnfunktion R(s) gibt die direkte Belohnung an, die der Agent erhält wenn er einen Zustand s erreicht.

Die Aktionen die der Agent bei erreichen des Ausgangs ausgeführt hat, werden im ersten Versuch und vielleicht in den Nachfolgenden Versuchen wahrscheinlich nicht optimal sein. Nicht optimal in dem Sinne, dass die Aktionssequenz nicht die kürzeste sein wird. Der Agent kann den Wert für die nummerische Belohnung maximieren, indem er eine optimale Strategie entwickelt, die den kürzesten Pfad findet. Eine optimale Strategie die für jeden möglichen Zustand eindeutig definiert, welche Aktion er durchführen muss, um so wenig wie möglich Aktionen zu verwenden und den Ausgang zu erreichen. Eine genauere Beschreibung der optimalen Strategie wird in den nachfolgenden Abschnitten gegeben.

3.3 Markov Entscheidungsprozess

Der Markov Entscheidungsprozess (MEP) oder MDP (engl. Markov decision process) nach Russell und Norvig [RN12, S. 752 ff.] ist ein sequentielles Entscheidungsproblem für eine vollständige beobachtbare, stochastische Umgebung mit einem Markov-Übergangsmodell und additiven Gewinnen. Der MEP besteht aus einem Satz von Zuständen (mit einem Anfangszustand s_0), einem Satz Actions(s) von Aktionen in jedem Zustand, einem Übergangsmodell P(s' | s, a) und einer Gewinnfunktion R(s).

Ein sequentielles Entscheidungsproblem ist ein wichtiges Anwendungsgebiet des verstärkenden Lernens. Bei diesen Problemen ist dem Agenten der direkte Nutzen des Aktionsergebnisses nicht bekannt. Erst nach einer Folge von Aktionen

wird dem Agenten eine Belohnung zugeteilt, z.B. ein Agent der das Schachspielen lernen soll, erhält keine direkte Belohnung nach den einzelnen Zügen. Erst am Ende einer Partie, wenn der König geschlagen ist, wird dem Agenten eine positive Verstärkung für einen Sieg oder eine negative Verstärkung für eine Niederlage zugeteilt (verspätetet Belohnung).

Vollständig beobachtbare Spiele sind z.B. Schach, Reversi, Tic Tac Toe, 4-Gewinnt und Dame, denn jeder Spieler kennt immer den kompletten Spielzustand. Vollständig beobachtbare Spiele werden auch als Spiele mit vollständiger Information bezeichnet. Viele Kartenspiele wie zum Beispiel Skat, sind nur teilweise beobachtbar, denn der Spieler kennt die Karten des Gegners nicht oder nur teilweise [Ert16, S. 114].

Ein stochastischer Übergang ist nur in einer nicht deterministischen Umgebung möglich. Reversi, Tic Tac Toe und Schach sind deterministische Strategiespiele, d.h. jeder Nachfolgezustand ist eindeutig definiert, eine Aktionssequenz führt also immer zum selben Ergebnis. Backgammon ist ein nichtdeterministisches Strategiespiel, in diesem werden stochastische Übergänge durch ein Würfelergebnis bestimmt, es ist also vorher nicht eindeutig welcher Nachfolgezustand durch eine Aktion eintreten wird.

Das Übergangsmodell beschreibt das Ergebnis jeder Aktion in jedem Zustand. Ist das Ergebnis stochastisch, bezeichnet P(s' | s, a) die Wahrscheinlichkeit, den Zustand s' zu erreichen, wenn die Aktion a im Zustand s ausgeführt wird. Handelt es sich um einen Markov-Übergang, dann ist die Wahrscheinlichkeit s' von s zu erreichen, nur von s abhängig und nicht vom Verlauf der vorherigen Zustände. Ganz ähnlich definiert Wolfgang Ertel die Markov-Entscheidungsprozesse [Ert16, S. 291]. Seine Agenten bzw. die Strategien der Agenten verwenden für die Bestimmung des nächsten Zustandes s_{t+1} nur Informationen über den aktuellen Zustand s_t und nicht über die Vorgeschichte. Dies ist gerechtfertigt, wenn die Belohnung einer Aktion nur von aktuellem Zustand und aktueller Aktion abhängt.

Additive Gewinne nach Russell und Norvig [RN12, S. 756] bestimmen über das zukunftsbezogene Verhalten des Agenten. Verwendet der Agent Additive Gewinne, dann bedeutet das für den Agenten, jeder Nutzen eines Zustandes in einer gewählten Zustandsfolge ist gleich Wertvoll. Zudem ist die Summe der Zustandsnutzen endlich, deshalb auch Modell des endlichen Horizonts. Der Nutzen einer Zustandsfolge ist wie folgt definiert:

$$U_h([s_0, s_1, s_2...]) = R(s_0) + R(s_1) + R(s_2) + ...$$

Additive Gewinne können nur bei Spielen verwendet werden, die früher oder später immer in einem Endzustand terminieren. Spiele die unter Umständen nicht immer einen Endzustand erreichen haben keinen endlichen Horizont, sondern einen unendlichen Horizont, für diese Spiele ist ein Modell mit einem endlichen Horizont unangemessen, denn wir wissen nicht wie Lang die Lebensdauer des Agenten ist [KLM96, S. 250]. Das Modell des endlichen Horizonts oder **verminderte Gewinne** unterscheiden sich von den Additiven Gewinnen durch einen Vermeidungsfaktor γ. Der Verminderungsfaktor schwächt Zustände in der Zukunft immer weiter ab, d.h. je weiter ein zustand in der Zukunft liegt, desto mehr wird er abgeschwächt. Der Nutzen für den ersten Zustand der Zustandsfolge wird nicht abgeschwächt. Ist γ gleich 1, sind die verminderten Gewinne gleich den additiven Gewinnen, die additiven Gewinne sind also ein Sonderfall der verminderten Gewinne.

$$U_h([s_0, s_1, s_2...]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ...$$

Halten wir fest: der Nutzen einer gegebenen Zustandsfolge ist die Summe der verminderten Gewinne, die während der Folge erhalten werden.

Anwendung des MEP auf Reversi und Tic Tac Toe. Beide Strategiespiele sind sequentielle Entscheidungsprobleme, denn die einzelnen Züge werden nicht direkt Belohnt, erst am Spielende wird ein Gewinner und ein Verlierer oder ein Unentschieden verkündet und der Agent erhält eine verspätete Verstärkung die er auf die Spielzugsequenz aufteilen muss (siehe nachfolgender Abschnitt **??** Temporale Differenz Lernen). Wie bereits erwähnt sind die beiden Strategiespiele vollständig beobachtbar und nicht stochastisch, somit sind sie deterministisch. Ein stochastisches Übergangsmodell für die Wahrscheinlichkeiten der Zustandsübergänge ist für Reversi und Tic Tac Toe nicht sinnvoll, da beide Spiele nicht vom Zufall abhängen und für jede Aktion in jedem Zustand nur ein einziger Zustandsübergang möglich ist. Wir werden in dieser Arbeit ausschließlich additive Gewinne für Reversi und Tic Tac Toe verwenden, da diese nach einer endlichen Anzahl von Aktionen immer in einem Endzustand terminieren. Später klären wir noch die Frage, ob wir überhaupt ein Übergangsmodell, für die Lernverfahren benötigen, denn es existieren sowohl modellbasierte Lernverfahren (Dynamische Programmierung, speziell Wert-Iteration), als auch modellfreie Lernverfahren (TD- und Q-Lernen).

3.4 Optimale Taktiken

Nach Russell und Norvig [RN12, S. 757 f.] beeinflusst eine Taktik oder Strategie das Verhalten des Agenten, d.h. sie empfiehlt welche Aktion der Agent in jedem Zustand ausführen soll. Aus Tradition wird beim verstärkenden Lernen eine Taktik mit dem Symbol π gekennzeichnet. Die Abbildung der Zustände auf Aktionen ist folgendermaßen definiert $\pi : S \to A$ oder $\pi(s) = a$. Abhängig von den Dimensionen der Umgebung existieren unterschiedlich viele Taktiken. Eine optimale Taktik wird bestimmt durch den erwarteten Nutzen bei Ausführung der Taktik π beginnend in einem Startzustand s:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]. \tag{3.1}$$

Eine optimale Taktik hat im Vergleich zu allen anderen möglichen Taktiken einen gleich hohen oder höheren erwarteten Nutzen. Eine solche optimale Taktik wird gekennzeichnet durch π_s^*:

$$\pi_s^* = \underset{\pi}{\operatorname{argmax}}\, U^{\pi}(s). \tag{3.2}$$

Es ist möglich, dass mehrere optimale Taktiken für ein Problem existieren. Russell und Norvig erklären, dass für eine optimale Strategie π_s^*, auch π^* geschrieben werden kann, denn wenn Taktik π_a^* optimal beim Beginn in a und Taktik π_b^* optimal beim Start in b sind und sie einen dritten Zustand c erreichen, gibt es keinen vernünftigen Grund, dass sie untereinander oder mit π_c^* nicht übereinkommen.

Mit diesen Definitionen ist der wahre Nutzen eines Zustands einfach $U^{\pi^*}(s)-$ d.h. die erwartete Summe verminderter Gewinne, wenn der Agent eine optimale Taktik ausführt. Wir schreiben dies als U(s). Russell und Norvig unterstreichen den Sachverhalt, dass die Funktionen U(s) und R(s) ganz unterschiedliche Quantitäten sind, denn R(s) gibt den "kurzfristigen" Gewinn, sich in s zu befinden an, wohingegen U(s) den "langfristigen" Gesamtgewinn ab s angibt.

3.5 Dynamische Programmierung und Wert-Iteration

Verwenden wir bereits vorhandenes Wissen über Strategien und speichern dieses in Zwischenergebnisse über Teile von Strategien, dann bezeichnen wir diese Vor-

gehensweise zur Lösung von Optimierungsproblemen als dynamische Programmierung. Diese Vorgehensweise wurde bereits 1957 von Richard Bellman beschrieben [Ert16, S. 293]. Verfahren welche kein Wissen über bereits vorhandene Strategien verwendet sind z.B. Minimax-Suche, Alpha-Beta-Suche und Iterativ vertiefende Tiefensuche.

Im vorherigen Abschnitt haben wir gezeigt (mittels der Ausführungen von Russell und Norvig), dass der Nutzen U, in einem Zustand s, unter Beachtung einer Strategie π, berechnet werden kann aus der Summe aller abgeschwächten Belohnungen, für jeden besuchten Zustand, in einem Zeitintervall von $t = 0$ bis ∞ (siehe 3.4 Optimale Taktiken, Gleichung für den erwarteten Nutzen 3.1). Dementsprechend gibt eine optimale Taktik $\pi^*(s)$ für jeden Zustand s den Nachfolgezustand mit dem größtmöglichen erwarteten Nutzen an:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s'|s,a)U(s'). \tag{3.3}$$

Daraus folgt, dass es eine direkte Beziehung zwischen dem Nutzen eines Zustandes und dem Nutzen seiner Nachbarn gibt: Der Nutzen eines Zustandes ist der unmittelbare Gewinn für diesen Zustand plus dem erwarteten verminderten Gewinn des nächsten Zustandes, vorausgesetzt, der Agent wählt die optimale Aktion. Das bedeutet, der Nutzen eines Zustandes ist gegeben durch:

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U(s'). \tag{3.4}$$

Diese Gleichung wird als Bellman-Gleichung bezeichnet, nach Richard Bellman(1957). Die Nutzen der Zustände - durch Gleichung 3.1 als die erwarteten Nutzen nachfolgender Zustandsfolgen definiert - sind Lösungen der Menge der Bellman-Gleichungen [RN12, S. 759].

Die aus der Bellman-Gleichung formulierbare rekursive Aktualisierungsregel, auch die Bellman-Aktualisierung genannt, ist Hauptbestandteil des Wert-Iteration Algorithmus. Wolfgang Ertel notiert diese Aktualisierungsregel wie folgt [Ert16, S. 294]:

$$\hat{V}(s) = \max_a [r(s,a) + \gamma \hat{V}(\delta(s,a))]. \tag{3.5}$$

Dahingegen notieren Russell und Norvig die Bellman-Aktualisierung etwas anders [RN12, S. 760]:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a)U_i(s'). \tag{3.6}$$

Betrachten wir jetzt die Äquivalenzen der beiden Gleichungen. Ertel bezeichnet den Iterationsschritt in einem Zustand s als $\hat{V}(s)$ und Russell und Norvig definieren den Nutzwert für den Zustand s bei der i-ten Iteration als $U_i(s)$ und den Iterationsschritt bezeichnen sie als U_{i+1}. Die Gewinnfunktionen R(s) und r(s,a) sind leicht Unterschiedlich. Funktion R(s) gibt den direkten Gewinn in einem Zustand s an und Funktion r(s,a) den Gewinn für eine Aktion die im Zustand s ausgeführt wird. Die Funktionen $\max_a$ und $\max_{a \in A(s)}$ berechnen die Aktion a mit dem höchsten erwarteten Nutzen. Das stochastische Modell der Welt wird durch die Funktionen $\delta(s, a)$ und $P(s'|s, a)$ beschrieben. Beide Funktionen bilden die Wahrscheinlichkeit ab, dass ein Zustand s' erreicht wird, wenn eine Aktion a in Zustand s ausgeführt wird.

Den wahren Nutzen haben wir definiert als die erwartete Summe verminderter Gewinne. Die Verminderung wird in beiden Gleichungen durch den Abschwächungsfaktor γ notiert. Die erwartete Summe verminderter Gewinne ist die Summe aller Iterationsschritte bis zur Konvergenz beider Gleichungen. Der rekursive Funktionsaufruf in der Aktualisierungsregel von Wolfgang Ertel $\hat{V}(\delta(s, a))$ übergibt dem nächsten Iterationsschritt den Zustand s', der zu einer von δ bzw. von der Umgebung festgelegten Wahrscheinlichkeit eintrifft. In der Aktualisierungsgleichung von Russell und Norvig wird dies durch die Kombination des stochastischen Modells P(s' | s,a) und dem rekursiven Funktionsaufruf $U_i(s')$ realisiert.

Ein Lernverfahren (z.B. die adaptive dynamische Programmierung) welches die Wert-Iteration nutzt, wird im Rahmen dieser Arbeit jedoch nicht implementiert. Die dynamische Programmierung und die Wert-Iteration sind sehr wichtige Ansätze für Lernverfahren und die nachfolgenden Lernverfahren sind teilweise sehr eng mit Lernverfahren verwandt, die Wert-Iteration verwenden.

3.6 Temporale Differenz Lernen

Bei dieser Lernmethode werden die Nutzen der beobachteten Zustände an die beobachteten Übergänge angepasst, sodass sie mit den Bedingungsgleichungen (siehe Bellman-Gleichung) übereinstimmen. Allgemeiner können wir sagen, wenn ein Übergang vom Zustand s in den Zustand s' stattfindet, wenden wir die folgende Aktualisierung mit $U^\pi(s)$ an [RN12, S. 966 f.]:

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha(R(s) + \gamma U^\pi(s') - U^\pi(s)). \tag{3.7}$$

Hier ist α der Lernratenparameter. Weil diese Aktualisierungsregel die Differenz

der Nutzen aufeinanderfolgender Zustände verwendet, wird sie auch häufig als TD-Gleichung (Temporale Differenz) bezeichnet. Der Lernratenparameter α gibt an, wie stark neue Nutzwerte die derzeitige Bewertungsfunktion anpassen können.

3.7 TD-Q-Lernen

Das TD-Q-Lernen ist eine Variante des TD-Lernens und wird auch als Q-Lernen bezeichnet. Die Aufgabe des TD-Q-Lernenden Agenten ist eine optimale Strategie zu entwickeln, er lernt nicht wie bei einer Wert-Iteration eine wahre Nutzenfunktion U(s), sondern eine Q-Funktion. Eine Q-Funktion ist eine Abbildung von Zustands/Aktions-Paaren auf Nutzwerte. Q-Werte sind wie folgt mit Nutzwerten verknüpf [RN12, S. 973]:

$$U(s) = \max_a Q(s, a). \tag{3.8}$$

Eine Nutzenfunktion U(s) ist abhängig von den abgeschwächten Nutzwerten aller nachfolgenden Zustände. Ein TD-Agent der eine Q-Funktion lernt, braucht weder für das Lernen noch die Aktionsauswahl ein Modell der Form P(s' | s,a). Aus diesem Grund sagt man auch, das Q-Lernen ist eine modellfreie Methode [RN12, S. 974].

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a)). \tag{3.9}$$

Was ist jedoch der Unterschied zwischen einer Belohnungsfunktion r(s, a) und einer Q-Funktion Q(s, a)? Die Funktion r(s, a) ist von der Umgebung definiert und kann vom Agenten nicht beeinflusst werden. Sollte diese Funktion dem Agenten eine numerische Verstärkung von -0,5 zuweisen, dann kann der Agent dies nicht ändern. Der Agent soll versuchen die Zusammenhänge der Zustands/Aktions-Paare zu lernen und Entscheidungen basierend auf seinen Lernerfahrungen zu treffen. Dies bezeichnen wir dann als Q-Lernen. Die vom Agenten gelernten zusammenhänge werden in Q-Werten gespeichert. Folglich wird in Q(s, a) oder Q[s, a] die gelernte Erfahrung des Agenten, für ein Zustand/Aktions-Paar, gespeichert.

Kapitel 4

Problemanalyse und Anforderungsdefinition

» *Das Spiel ist die höchste Form der Forschung.* «
(Albert Einstein)

In Abschnitt 4.1 wird die Aufgabenstellung genauer analysiert, die Spieltheoretischen Verfahren und die Verfahren des maschinellen Lernens aus dem Grundlagenkapitel werden auf die beiden Strategiespiele Reversi und Tic Tic Toe angewendet. Abschnitt 4.2 definiert die Anforderungen die der Softwareprototyp erfüllen sollte. Die Anforderungen beziehen sich auf die Problematik und die Grundlagen.

4.1 Die Problematik

Das Thema der Arbeit ist Untersuchung der Lernfähigkeit verschiedener Verfahren am Beispiel von Computerspielen. Bevor die Lernfähigkeit der Verfahren untersucht werden kann, müssen wir die Computerspiele festlegen und analysieren. Wie bereits erwähnt werden wir die Lernfähigkeit der Verfahren am Beispiel der Strategiespiele Reversi und Tic Tac Toe untersuchen (siehe Abbildung 4.1). Eine genaue Beschreibung der Spielregeln, der Siegesbedingungen und möglicher Strategien bezüglich Heuristiken, wird in Kapitel 5 Modellierung und Entwurf erfolgen. Die zentrale Frage ist: wie kann ein Programm lernen ein Computerspiel erfolgreich zu spielen?

Die Spieltheorie aus Abschnitt **??** liefert gleich mehrere Ansätze diese Frage zu beantworten. Die kombinatorische Suche (Abschnitt 2.3.1 Minimax) probiert einfach alle Möglichkeiten aus und liefert die beste gefundene Möglichkeit zurück. Die reine kombinatorische Minimax Suche ist praktisch jedoch nicht anwendbar, da, wie bereits im Abschnitt Minimax beschrieben wurde, die Anzahl der Kom-

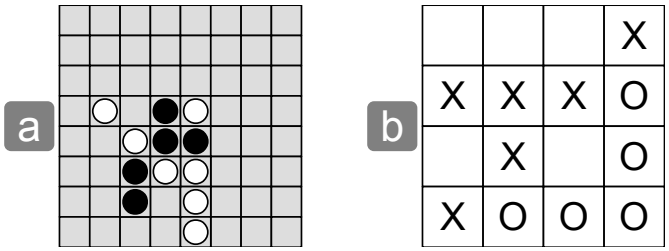

Abbildung 4.1 Tic Tac Toe und Reversi Spielzustände.

binationsmöglichkeiten mit der Komplexität des Ausgangsproblems exponentiell ansteigt.

Selbst mit einer Kürzung von ganzen Unterbäumen des Suchbaums, ist die Rechenzeit für realistische Probleme nicht handhabbar (siehe Abschnitt 2.3.2 Alpha-Beta-Kürzung). Das Kürzen des Suchbaums kann unter Umständen mit einer iterativ vertiefenden Tiefensuche verbessert werden (siehe Abschnitt 2.3.3 Iterativ vertiefende Tiefensuche). Die iterativ vertiefende Suche könnte Züge, z.B. in einer Tiefe von 2, sortieren. Vielversprechende Spielzüge könnten zu erst ausprobiert werden und das Alpha-Beta Verfahren könnte einen größeren Teil des Suchbaums kürzen.

Eine weitere Möglichkeit die Suche nach dem optimalen Spielzug in jeder Spielsituation zu verbessern, ist das Vermeiden von Übergängen. Ein Übergang oder Transition ist ein Spielzustand der mehrfach, an verschiedenen Stellen, in einem Suchbaum auftreten kann. Übergangstabellen und Transitions sind ausführlich in Abschnitt 2.3.4 Übergangstabellen erläutert. Eine Vermeidung dieser Übergänge könnte eine weitere Rechenzeitverringerung bewirken.

Die Heuristik oder Bewertungsfunktion ist wohl der wichtigste Leistungsfaktor aus den Verfahren der Spieltheorie (siehe Abschnitt 2.3.5 Heuristik). Eine Heuristik kann jeden Knoten des Suchbaums bewerten (Nutzwert) und nicht nur die Blätter. Dies ermöglicht es die Suche nach einer bestimmten Zeitspanne oder iterierten Tiefe abzubrechen und den Spielzustand mit der besten Bewertung zurück zu geben.

Die Vorteile einer Heuristik sind die Limitierung der Zeit, die für eine Suche benötigt wird und dass das bisher beste gefundene Ergebnis zurück gegeben wird. Der große Nachteil einer Bewertungsfunktion ist, ein ermittelter Nutzwert für einen Spielzustand kann falsch sein. Die Qualität einer Heuristik ist also ausschlaggebend für die Spielstärke des Programms. Heuristiken sind zudem stark abhängig von ihrer Spielgrundlage, d.h. sowohl Reversi als auch Tic Tac Toe benötigen eigene Heuristiken, die individuell den Nutzen der Stellungen bewerten.

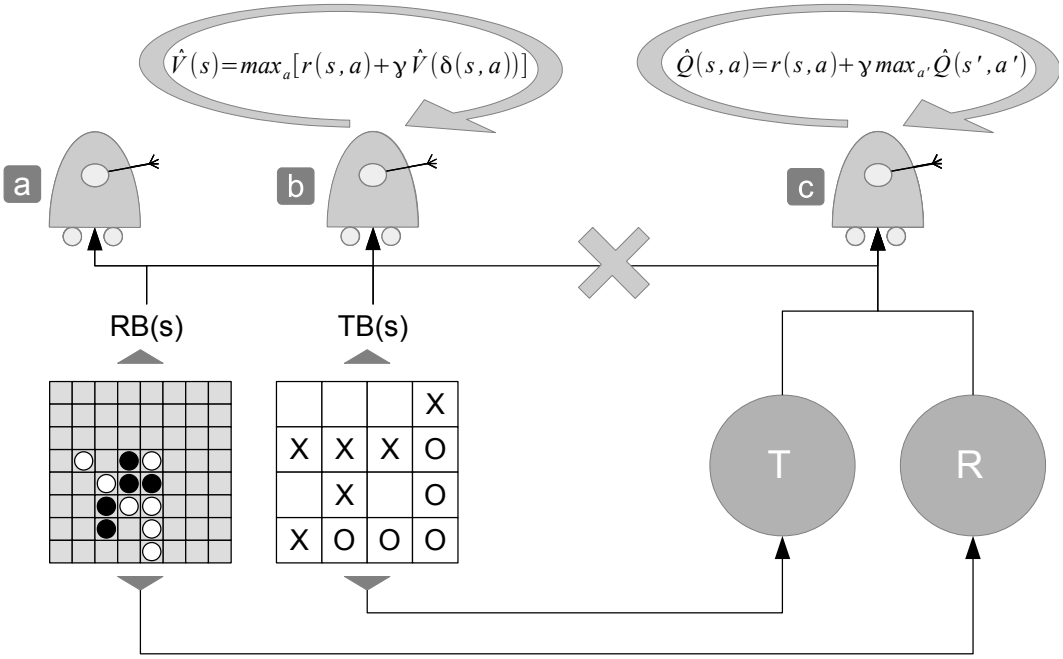

Abbildung 4.2 Die Projektproblematik.

Die drei Agenten aus Abbildung 4.2 repräsentieren drei Programme, die mit unterschiedlichen Verfahren, die selbe Aufgabe lösen. Die Aufgabe lautet: für jeden Spielzustand der Strategiespiele Reversi und Tic Tac Toe sollen die Agenten (a), (b) und (c) einen optimalen Spielzug (eine Aktion) vorschlagen, d.h. die Agenten müssen eine optimale Strategie anwenden. Eine optimale Strategie verwendet für jeden Möglichen Spielzustand immer den bestmöglichen Spielzug.

Agent (a) ist ein nicht lernender Agent, der Verfahren aus der Spieltheorie anwendet und dem die Bewertungsfunkionen RB(s) und TB(s) zur Verfügung stehen. RB(s) ist die Reversi Bewertungsfunktion, mit einem Reversi Spielzustand s als Eingabeparameter. TB(s) ist die Tic Tac Toe Bewertungsfunktion, mit einem Tic Tac Toe Spielzustand s als Eingabeparameter.

Agent (b) ist ein lernender Agent, der wie der Agenten (a) über die Bewertungsfunktionen RB(s) und TB(s) verfügt. Dieser Agent soll die Gewichtungen (Faktoren a_x) der Bewertungsfunktionen mittels Spielerfahrung lernen. Er spielt eine festgelegte Anzahl von Spielen z.B. gegen sich selbst und verwendet eine Aktualisierungsfunktion $\hat{V}(s,a)$. Diese Funktion wurde bereits in Abschnitt **??** Wert-Iteration

und Dynamische Programmierung behandelt. Zusammengefasst ist diese Funktion eine Iterationsvorschrift, die mit der Bellman-Rekursionsgleichung die Gewichtungen der Heuristik so lange anpasst, bis diese sich nicht weiter verändern lassen und gegen eine optimale Strategie konvergieren.

Agent (c) ist ein lernender Agent, der im Gegensatz zu den Agenten (a) und (b) über kein Modell der Spielwelt verfügt, also ist ihm nicht bekannt in welchen Spielzustand ihn seine Aktionen führen. Er erhält auch keine Bewertungsfunktionen RB(s) und TB(s) für Spielzustände s. Alles was der Agent erhält sind zwei Menge T und R. Menge T enthält alle möglichen Aktionen die der Agent in einer bestimmten Tic Tac Toe Spielsituation ausführen darf, äquivalent dazu enthält Menge R alle möglichen Aktionen die der Agent in einer bestimmten Reversi Spielsituation ausführen darf. Nach einer bestimmten Anzahl von Aktionen erhält der Agent verspätet unterschiedliche Belohnungen für einen Sieg, eine Niederlage oder ein Unentschieden des Spiels. Der Agent verwendet das Lernverfahren $\hat{Q}$(s, a) aus Abschnitt **??**, um eine optimale Strategie zu lernen.

Sind alle Agenten implementiert, können wir beginnen die Lernfähigkeit der Agenten zu vergleichen. Wie werden die beiden lernenden Agenten gegen den nicht lernenden Agenten abschneiden? Welcher Agent wird der, der am meisten gewinnt und welcher Agent wird am meisten Verlieren? Wie viel Zeit benötigen die Agenten für die Berechnung der optimalen Strategie für die beiden Strategiespiele (Training)? Diese Fragen werden in Kapitel **??** Auswertung beantwortet.

4.2 Anforderungen

Im nachfolgenden Abschnitt definieren wir die funktionalen Anforderungen der Software. Wir bestimmten, welche Funktionalitäten die Strategiespiele und die Agenten mindestens haben und wie die Agenten getestet werden sollen. Wir definieren die Funktionalitäten, um den Funktionsbereich der Software einzugrenzen und einen Überblick zu verschaffen.

4.2.1 Tic Tac Toe Spielumgebung

Die Spielumgebung soll die in Abschnitt 2.1 definierten Tic Tac Toe Spielregeln implementieren. Die Tic Tac Toe Spielumgebung repräsentiert eine Testumgebung für die Agenten, der Zufallsagent wird in dieser Umgebung gegen den TicTacToe-Heuristik Agenten antreten. Der TD-Q-Lernende Agent soll zuerst diese Umgebung erkunden und lernen sich in der Umgebung zurecht zu finden, d.h. der TD-Q-Lernende Agent soll eine TicTacToe-Siegesstrategie entwickeln.

makeMove(position):
Die Funktion soll Koordinaten erhalten. Die Koordinaten definiert exakt, auf welches Spielfeld eine Spielfigur gesetzt werden soll. Die Funktion soll diesen Spielzug, sollte dieser Regelkonform sein, ausführen.

undoMove():
Die Funktion soll den letzten durchgeführten Spielzug revidieren.

getPossibleMoves(): return list
Die Funktion soll eine Liste von Koordinaten liefern. In dieser Liste sind nur mögliche und regelkonforme Spielzüge (Koordinaten) enthalten.

isTerminal: return bool
Die Funktion soll True zurück liefern, wenn der aktuelle Zustand der Umgebung ein Endzustand (Terminalzustand) ist, andernfalls liefert die Funktion ein False.

getReward: return float
Die Funktion soll eine nummerische Belohnung liefern. Die Belohnung soll abhängig sein vom aktuellen Spielzustand.

4.2.2 Reversi Spielumgebung

Die Spielumgebung soll die in Abschnitt 2.2 definierten Reversi Spielregeln implementieren. Die Reversi Spielumgebung repräsentiert eine Testumgebung für die Agenten, der Zufallsagent wird in dieser Umgebung gegen den Reversi-Heuristik Agenten antreten. Der TD-Q-Lernende Agent soll zuerst diese Umgebung erkunden und lernen sich in der Umgebung zurecht zu finden, d.h. der TD-Q-Lernende Agent soll eine Reversi-Siegesstrategie entwickeln.

makeMove(position):
Die Funktion soll Koordinaten erhalten. Die Koordinaten definiert exakt, auf welches Spielfeld eine Spielfigur gesetzt werden soll. Die Funktion soll diesen Spielzug, sollte dieser Regelkonform sein, ausführen.

undoMove():
Die Funktion soll den letzten durchgeführten Spielzug revidieren.

getPossibleMoves(): return list
Die Funktion soll eine Liste von Koordinaten liefern. In dieser Liste sind nur mögliche und regelkonforme Spielzüge (Koordinaten) enthalten.

isTerminal: return bool
Die Funktion soll True zurück liefern, wenn der aktuelle Zustand der Umgebung ein Endzustand (Terminalzustand) ist, andernfalls liefert die Funktion ein False.

getReward: return float
Die Funktion soll eine nummerische Belohnung liefern. Die Belohnung soll abhängig sein vom aktuellen Spielzustand.

4.2.3 Agent des Zufalls

Der Agent des Zufalls soll den schlechtesten Spieler symbolisieren. Er soll seine Entscheidungen vollkommen zufällig treffen. In Kapitel 7 Validierung werden wir diesen Agenten, als Gegenspieler für die Heuristik Agenten und die lernenden TD-Q-Agenten einsetzen.

suggestRandomTicTacToeAction(ticTacToeState): return tuple
Diese Funktion soll eine Tic Tac Toe Spielsituation übergeben bekommen, d.h. eine Instanz der TicTacToe Klasse. Die Funktion soll eine zufällige, aber zulässige, Aktion zurückgeben.

suggestRandomReversiAction(reversiState): return tuple
Diese Funktion soll eine Reversi Spielsituation übergeben bekommen, d.h. eine Instanz der Reversi Klasse. Die Funktion soll eine zufällige, aber zulässige, Aktion zurückgeben.

4.2.4 Tic Tac Toe Heuristik Agent

Der Agent soll die in Abschnitt 2.3.5 erstellte Tic Tac Toe Heuristik und eine 2-Spielzüge vorausschauende Alpha-Beta Suche verwenden (siehe Abschnitt 2.3.3 und 2.3.2). Dieser Agent soll einen fortgeschrittenen Spielgegner repräsentiert, d.h. wir müssen mittels Testspielen gegen den Zufallsagenten zeigen, dass der Tic Tac Toe Heuristik Agent verhältnismäßig oft gewinnt. Dieser Agent soll in Tic Tac Toe Testspielen gegen den TD-Q-Agenten antreten. Die Ergebnisse sollen dabei helfen, die Leistungsfähigkeit und Grenzen des TD-Q-Lernens, hinsichtlich dem Lernen von Tic Tac Toe, zu beurteilen.

suggestAction(ticTacToeState): return tuple
Diese Funktion soll eine Tic Tac Toe Spielsituation übergeben bekommen. Die Funktion soll, abhängig von der erhaltenen Spielsituation, eine Aktion vorschlagen. Die

Aktion soll mittels der TicTacToe-Heuristik und eienr 2-Zug Vorausschau und Alpha-Beta-Suche ermittelt werden.

4.2.5 Reversi Heuristik Agent

Der Agent soll die in Abschnitt 2.3.5 erstellte Reversi Heuristik und eine 2-Spielzüge vorausschauende Alpha-Beta Suche verwenden (siehe Abschnitt 2.3.3 und 2.3.2). Dieser Agent soll einen fortgeschrittenen Spielgegner repräsentiert, d.h. wir müssen mittels Testspielen gegen den Zufallsagenten zeigen, dass der Reversi-Heuristik Agent verhältnismäßig oft gewinnt. Dieser Agent soll in Reversi Testspielen gegen den TD-Q-Agenten antreten. Die Ergebnisse sollen dabei helfen, die Leistungsfähigkeit und Grenzen des TD-Q-Lernens, hinsichtlich dem Lernen von Reversi, zu beurteilen.

suggestAction(reversiState): return tuple
Diese Funktion soll eine Reversi Spielsituation übergeben bekommen. Die Funktion soll, abhängig von der erhaltenen Spielsituation, eine Aktion vorschlagen. Die Aktion soll mittels der Reversi Heuristik und eienr 2-Zug Vorausschau und Alpha-Beta-Suche ermittelt werden.

4.2.6 Tic Tac Toe TD-Q lernender Agent

Der Agent soll, mittels des in Abschnitt 3.7 behandelten TD-Q-Lernens, eine Siegesstrategie für das Strategiespiel Tic Tac Toe entwickeln. Testspiele gegen den Zufallsagenten und den Tic Tac Toe Heuristik Agenten, sollen eine Untersuchung der Leistungsfähigkeit und der Grenzen des TD-Q-Lernens ermöglichen.

learnTicTacToeInXGames(amountOfGames):
Die Funktion soll den Lernmodus des Agenten realisieren. Der Eingabeparameter legt die Anzahl der Trainingsspiele fest. Die Lernerfahrungen während dieser Trainingsspiele, sollen in einer SQLite Datenbank gespeichert werden.

suggestAction(ticTacToeState): return tuple
Die Funktion soll eine Tic Tac Toe Spielsituation übergeben bekommen. Ausgehend von der Eingangsspielsituation, ist nur eine bestimmte Anzahl von Aktionen möglich. Abhängig von seinen Erfahrungen und dem gegebenen Spielzustand, soll der Agent die mögliche Aktion mit dem höchsten gelernten Q-Wert zurückgeben.

4.2.7 Reversi TD-Q lernender Agent

Der Agent soll, mittels des in Abschnitt 3.7 behandelten TD-Q-Lernens, eine Siegesstrategie für das Strategiespiel Reversi entwickeln. Testspiele gegen den Zufallsagenten und den Reversi Heuristik Agenten, sollen eine Untersuchung der Leistungsfähigkeit und der Grenzen des TD-Q-Lernens ermöglichen.

learnReversiInXGames(amountOfGames):
Die Funktion soll den Lernmodus des Agenten realisieren. Der Eingabeparameter legt die Anzahl der Trainingsspiele fest. Die Lernerfahrungen während dieser Trainingsspiele, sollen in einer SQLite Datenbank gespeichert werden.

suggestAction(reversiState): return tuple
Die Funktion soll eine Reversi Spielsituation übergeben bekommen. Ausgehend von der Eingangsspielsituation, ist nur eine bestimmte Anzahl von Aktionen möglich. Abhängig von seinen Erfahrungen und dem gegebenen Spielzustand, soll der Agent die mögliche Aktion mit dem höchsten gelernten Q-Wert zurückgeben.

4.2.8 Testen der Agenten

Eine Testumgebung, in der alle Agenten gegeneinander Spielen. Der Zufallsagent soll in 100 Testspielen gegen den Tic Tac Toe Heuristik Agenten, den Reversi Heuristik Agenten, den Tic Tac Toe TD-Q-Lernen Agenten und den Reversi TD-Q-Lernen Agenten antreten. Der Tic Tac Toe Heuristik Agent soll 100 Testspiele gegen die drei Lernstadien des Tic Tac Toe TD-Q-Lernen Agenten spielen. Im ersten Lernstadium soll der TD-Q-Lernen Agent, in 100 Trainingsspielen gegen sich selbst, eine Strategie entwickeln. Im zweiten Lernstadium sollen es 1000 und im dritten Lernstadium 10000 Trainingsspiele sein. Äquivalent gilt dies auch für den Reversi TD-Q-Lernen Agenten und den Reversi Heuristik Agenten. Selbstverständlich spielen die Reversi Agenten, in der Reversi Spielumgebung und die Tic Tac Toe Agenten, in der Tic Tac Toe Spielumgebung.

Kapitel 5

Modellierung und Entwurf

In diesem Kapitel werden die funktionalen Anforderungen aus dem Abschnitt **??** spezifiziert. Modelle für die einzelnen funktionalen Anforderungen sollen entwickelt werden. Die Modelle veranschaulichen das geforderte funktionale Verhalten der Software. Voraussetzung für die Entwicklung der Modelle ist die Konkretisierung der funktionalen Anforderungen. Diese Konkretisierung beinhaltet die Festlegung der Bestandteile die für eine Implementierung der funktionalen Anforderung benötigt werden. Ziel des Kapitels ist es, die wichtigsten Bestandteile einer funktionalen Anwendung herauszubilden, dieses Bestandteile zu definieren und zu veranschaulichen.

5.1 Tic Tac Toe Heuristik

Die nachfolgend aufgestellte Strategie ist keine optimale Strategie, das heißt es ist nicht möglich mit dieser Strategie immer zu gewinnen oder mindestens ein Unentschieden zu erreichen. Das Ziel des Lernverfahrens ist es, diese nicht optimale Strategie in eine optimale Strategie oder zumindest eine annähernd optimale Strategie zu transformieren. Eine optimale Strategie würde gegen unaufmerksame oder unerfahrene Gegner verhältnismäßig oft Gewinnen und nicht gegen diese Verlieren, Unentschieden können trotzdem vorkommen. Ist der Gegner ein perfekter TicTacToe Algorithmus oder ein TicTacToe Großmeister, dann sollte die optimale Strategie überwiegend Unentschieden hervorbringen. Eine optimale Strategie sollte in 100 Spielen gegen einen Großmeister oder eine andere optimale Strategie (dieselbe optimale Strategie oder möglicherweise eine andere) 100 Unentschieden erringen. Siege sind theoretisch höherwertiger als Unentschieden, aber innerhalb der TicTacToe Spielwelt sind diese gegen einen Großmeister oder einen perfekten Algorithmus äußerst unwahrscheinlich.

Unser Lernalgorithmus oder im Kontext des verstärkenden Lernens unser Agent, erhält eine Belohnung von +1 wenn er eine Party (eine komplette Spielzugsequenz

bis ein Spielergebnis feststeht) TicTacToe gewinnt. Verliert er eine Party, dann wird er bestraft mit dem numerischen Wert -1. Bei einem Unentschieden wird der Agent ebenfalls belohnt, aber die Belohnung ist nicht so hoch wie bei einem Sieg, denn wir wollen das Verhalten des Lernverfahrens so trainieren, dass es eher einen Sieg erlangen wird als ein Unentschieden, aber auf jeden Fall eine Niederlage vermeidet. Der Agent soll immer versuchen diesen nummerischen Wert zu maximieren, darum wird er eine Niederlage vermeiden. Der Agenten könnte auch so trainiert werden, dass er immer absichtlich verlieren würde, dafür müsste jeder, vom Agenten ausgeführte, Spielzug (egal welcher Spielzug) eine hohe negative nummerische Bestrafung hervorbringen. Das Ziel des Agenten wäre dann, schnellstmöglich ein Ende des Spiels zu provozieren und weil verlieren sicherer und kürzer ist als gewinnen oder ein Unentschieden, würde der Agent lernen absichtlich zu verlieren. Interessant wäre das Spielergebnis, wenn zwei Agenten gegeneinander antreten würden, die immer schnellstmöglich verlieren wollen, dann entstünden theoretisch ausschließlich Unentschieden.

Das Spielfeld

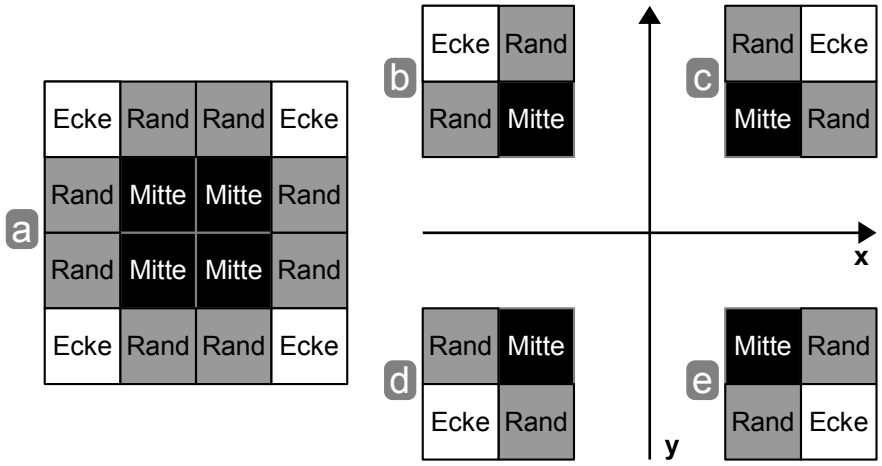

Abbildung 5.1 Symmetrie Eigenschaften des vier mal vier Tic Tac Toe Spielfelds.

Um eine Strategie zu entwickeln werden wir uns zuerst das Spielfeld ansehen und dieses analysieren. Das vier mal vier TicTacToe Spielfeld hat 16 Spielfelder, vier Eckfeldern, vier Mittelfeldern und acht Randfeldern (siehe Abbildung 5.1 a). Ziehen wir eine horizontale und eine vertikale Achse durch das Spielfeld, dann

sind bestimmte Symmetrieeigenschaften zu erkennen, Abbildung 5.1 zeigt b und c, sowie d und e sind symmetrisch zur y-Achse, b und d, sowie c und e sind symmetrisch zur x-Achse und b und e, sowie d und c sind symmetrisch, wenn man sie an der x-Achse und der y-Achse spiegelt. Diese Symmetrieeigenschaften sind wichtig für die Reduktion der Strategien, das heißt wenn eine Strategie auf b angewendet werden kann, dann auch auf c, d und e.

Kontrolliere die Mitte

Um über bestimmte Spielfelder reden zu können, müssen wir eine konkrete Identifikation der einzelnen Spielfelder vornehmen. In Abbildung 5.2 wird daher jedem der Spielfelder ein Index zugewiesen.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)

Abbildung 5.2 Die Indizes der einzelnen Spielfelder.

Die Eröffnungsstrategie konzentriert sich auf die Kontrolle der Mittelfelder. Hat der Agent beziehungsweise das Lernverfahren das Recht auf den ersten Zug befindet er sich immer in Zustand s_0. In diesem Zustand sind alle Spielfelder leer und der Agent kann durch Zufall eines der vier mittleren Felder wählen. Der Agent trifft diese Entscheidung zufällig, weil zu diesem Zeitpunkt und in diesem Zustand alle vier Mittelfelder die gleiche positive numerische Belohnung erbringen (siehe Abbildung 5.3 1). Die Rand- und Eckfelder haben keine numerische Belohnung, aber auch keine numerische Bestrafung für den Agenten, so wird sichergestellt, dass das Lernverfahren sich für ein mittleres Spielfeld entscheidet. Nachdem der gegnerische Spieler seine Spielfigur gesetzt hat, soll der Agent die zweite Spielfigur ebenfalls auf ein mittleres Feld setzen, jedoch eher auf ein mittleres Spielfeld, welches sich in einer Reihe ohne eine gegnerische Spielfigur befindet.

Von jetzt an betrachten wir die beiden Kontrahenten Alice (Spielfiguren X) und Bob (Spielfiguren O) als zwei Instanzen des selben Lernverfahrens, dass heißt der Agent spielt gegen einen anderen Agenten mit exakt dem selben Verhalten. In Abbildung 5.3 (2a) setzt Alice ihre Spielfigur auf das Feld (2,1) und Bob auf (1,2).

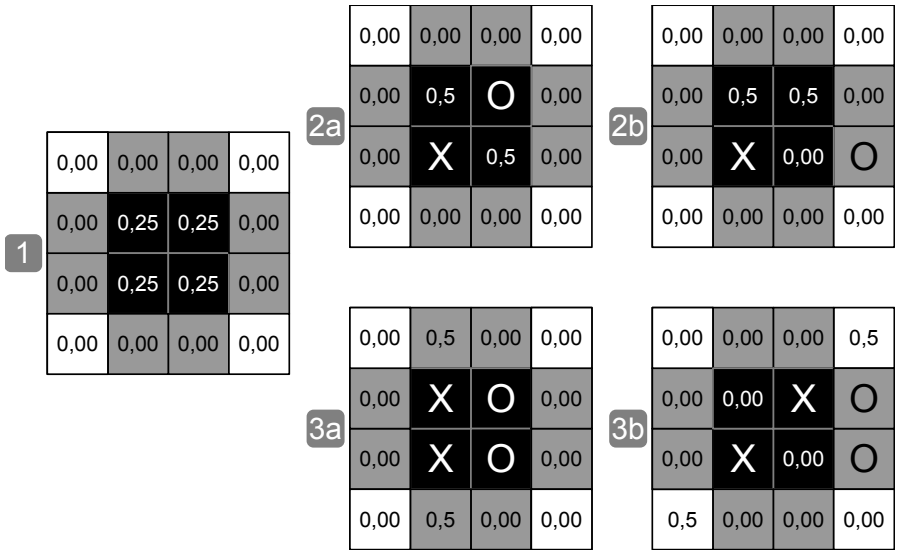

Abbildung 5.3 Strategie um die Mitte zu kontrollieren.

Die Strategie der Mittelfeld Kontrolle offenbart unserer Agentin Alice zwei Aktionsmöglichkeiten mit positiver Belohnung. Setzt Agentin Alice ihre Spielfigur auf die Mittelfelder (1,1) oder (2,2), dann erhält sie einen numerische Belohnung von +0,5. Alice entscheidet durch Zufall welches der beiden Felder mit der gleichgroßen größtmöglichen Belohnung sie auswählt. Alice entscheidet sich durch Zufall für das Spielfeld (1,1) und Bob, der ebenfalls der Strategie der Mittelfeld Kontrolle folgt, entscheidet sich für das letzte nicht besetzte Mittelfeld (Abbildung 5.3 3a).

Sollte Agent Bob eine andere gelernte Strategie verfolgen und zum Beispiel ein Randfeld besetzen, dann werden Agentin Alice andere Aktionen von ihrer Strategie Vorgeschlagen. In einem alternativen Spielverlauf (Abbildung 5.3 2b) setzt Agent Bob seine Spielfigur nicht auf das Spielfeld (1,2), sondern auf das Spielfeld (2,3). Die Strategie der Kontrolle der Mittelfelder bevorzugt Mittelfelder, die in einer leeren vertikalen, horizontalen oder diagonalen Reihe sind, bezogen auf die bereits gesetzte Spielfigur. Das Mittelfeld (2,2) wird uninteressant für die Strategie von Alice, weil eine mögliche horizontale Reihe aus vier gleichen Spielsteinen bereits nicht mehr möglich ist. Dahingegen schlägt die Strategie die Mittelfelder (1,1) und (1,2) mit einer numerischen Belohnung von +0,5 vor. Setzt Alice ihre Spielfigur auf das Mittelfeld (1,1), dann ist eine vertikale Verbindung von vier gleichen Spielfiguren möglich und setzt Alice ihre Spielfigur auf das Mittelfeld (1,2), dann ist eine diagonale Verbindung von vier gleichen Spielfiguren möglich.

Agentin Alice entschiedet durch Zufall ihre Spielfigur auf das Mittelfeld (1,2) zu setzen, denn der Agent soll durch Zufall entscheiden, wenn für mehrere Aktionen

in einem Zustand die gleich Hohe größtmögliche Belohnung vergeben wird. Agent Bob setzt daraufhin seine Spielfigur auf das Randfeld (1,3). Ein neue Spielsituation (Zustand) entsteht (Abbildung 5.3 3b). Alice erhält von der Strategie der Mittelfeld Kontrolle die Aktionsoptionen (0,3) und (3,0). Beide Aktionsoptionen werden mit +0,5 belohnt. Diese Strategie berücksichtigt nicht, dass Agent Bob bereits zwei Spielsteine in einer ungestörten vertikalen Verbindung positioniert hat. Daher kann Alice durch Zufall entscheiden welche Aktion sie ausführt. Eine bessere Strategie würde Alice in diesem Zustand diese Option nicht lassen, denn das Eckfeld (0,3) ist in dieser Spielsituation attraktiver als das Eckfeld (3,0). Eckfeld (0,3) erweitert die diagonale ungestörte Verbindung von Agentin Alice auf eine Länge von drei und gleichzeitig würde die vertikale ungestörte Verbindung von Agent Bob gestört werden. Eine optimale Strategie würde Alice mit einer Belohnung von +0,75 das Eckfeld (0,3) empfehlen.

Bevor wir Alice und Bob die Möglichkeit geben die Strategie selber zu verbessern beziehungsweise eigenständig zu entwickeln und solche Auffälligkeiten und Muster in die Strategie zu integrieren, werden wir die Ausgangsstrategie noch um eine Teilstrategie erweitern.

Verteidigung ist der beste Angriff

Diese Strategie konzentriert sich darauf, gegnerische Stellungen zu erkennen und dahingehend Gengenmaßnahmen einzuleiten. Abbildung 5.4 verdeutlicht gefährliche Spielsituationen, in denen die Strategie Gegenmaßnahmen vorschlagen sollte. Horizontale Verbindugsmöglichkeiten (1a - 1d)

5.2 Reversi Heuristik

5.3 Die Strategiespielumgebungen Tic Tac Toe und Reversi

getPossibleActions()

Berechnen eine Liste von Positionskoordinaten in der Form:

$$[Tupel(X_a Koordinate, Y_b Koordinate), ...].$$

Diese Koordinaten geben die Spielfelder an, welche in einem korrekten Spielzug, von dem aktuellen Spieler, belegt werden können. Aus dieser Liste von Aktionen

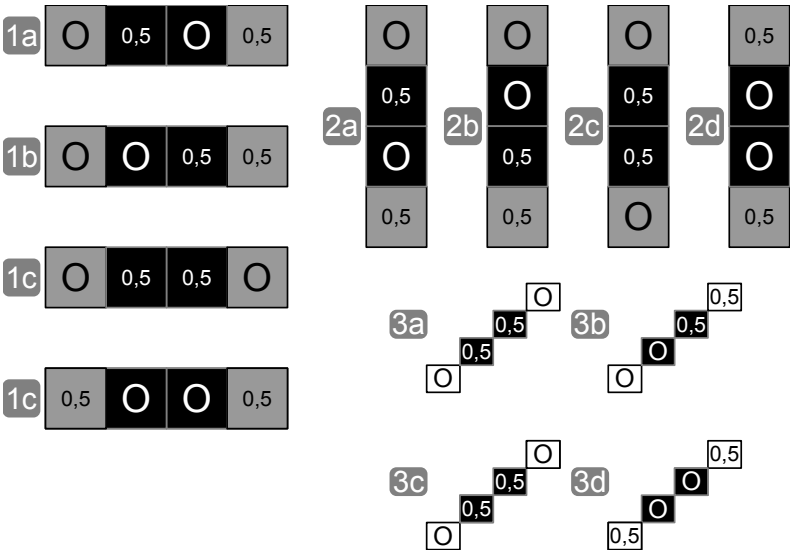

Abbildung 5.4 TicTacToe Angriffsstrategien (aus der Sicht von Bob) und Verteidigungsstrategien (aus der Sicht von Alice).

wählt der Spieler oder der Agent eine mögliche Aktion aus.

5.4 Agent ohne Lernen

Der Agent ohne Lernen bedient sich diverser Spieltheoretischer Verfahren aus dem Grundlagenkapitel **??** Spiele mit Gegner. Auf der Basis des alpha-beta gekürzten Minimax Suchbaumverfahrens versucht der Agent bis zu einer festgelegten Suchtiefe ein optimales Ergebnis zu finden. Die festgelegte Suchtiefe wurde aus dem Abschnitt 2.3.3 Iterativ vertiefende Tiefensuche in den Algorithmus des nicht lernenden Agenten übernommen. Diese maximale Suchtiefe begrenzt die Rechenzeit des Agenten, sodass der Agent beim explorieren und expandieren des Suchbaums irgendwann abbricht und das bisher beste gefundene Ergebnis zurück gibt.

Diese Rechenzeitverbesserung ist nur durch die Verwendung einer Heuristik möglich. Wie bereits in Abschnitt 2.3.5 Heuristik erklärt, ermöglicht es die Heuristik oder Bewertungsfunktion auch nicht Blattknoten des Suchbaums zu evaluieren, sprich den Nutzen von nicht Endzuständen und Endzuständen zu berechnen. Die Qualität der Heuristik ist ausschlaggebend für die Qualität (Spielstärke) dieses nicht lernenden Agenten. Der nicht lernende Agent hat keine Möglichkeit diese Heuristik anzupassen. Dahingegen kann der Agent mit TD-Lernen die Parameter der Ausgangsheuristik, anhand seiner Spielerfahrung, anpassen. Der Agent ohne Lernen bietet folgende Funktionalitäten:

getStrategicTicTacToeAction(ticTacToeGameState)
Der Input dieser Funktion ist ein Tic Tac Toe Objekt, welches den aktuellen Spielstatus repräsentiert. Wie bereits erklärt sucht diese Funktion mittels Alpha-Beta-Suche bis zu einer begrenzten Suchtiefe nach dem bestmöglichen Zustandsnutzen. Dieser Zustandsnutzen wird durch die in Abschnitt 5.1 entworfene Tic Tac Toe Heuristik berechnet. Der zurückgegebene Wert ist die Aktion, welche die erste Aktion des Pfades (Aktionssequenz) zum gefundenen bestmöglichen Zustandsnutzen ist.

getStrategicReversiAction(reversiGameState)
Diese Funktion ist der Funktion getStrategicTicTacToeAction(ticTacToeGameState) sehr ähnlich, nur dass diese auf die Reversi Strategiespielwelt angewendet wird. Der Input der Funktion ist somit ein Reversi Objekt und die verwendete Heuristik ist ebenfalls auf Reversi zugeschnitten(siehe Abschnitt 5.2). Der Output ist die vorgeschlagene bestmögliche Aktion, im aktuellen Spielzustand, hinsichtlich der abgeschnittenen Suche und der Reversi Bewertungsfunktion.

Alternative Realisierung wäre z.B. die Aufteilung des Agenten ohne Lernen in zwei Agentenklassen. Ein Agent für Tic Tac Toe und ein Agent für Reversi. Beide Agenten ohne Lernen würden dann über eine Funktion getStrategicAction(gameState) verfügen. Eine andere alternative wäre die Übergabe der Heuristiken an die Funktionen. In einer Programmiersprache in der Funktionen einer höheren Ordnung (Higher-order functions) erlaubt sind, könnten die Heuristiken für Reversi und Tic Tac Toe als Eingabeparameter übergeben werden. Auf diese Weise müsste nur ein nicht lernender Agent und eine Funktion getStrategicAction(gameState, heuristicFunction) implementiert werden. Wir wollen die funktionale Programmierung in dieser Arbeit jedoch nicht anwenden, sondern nur eine mögliche alternative Implementierung aufzeigen. Diese beiden alternativen Realisierungen können auch auf die lernenden Agentenmodelle übertragen werden.

5.5 Agent mit TD-Q-Lernen

Kapitel 6

Algorithmen und Implementierung

In diesem Kapitel: //TODO Einführung in das Kapitel

6.1 Tic Tac Toe

6.2 Reversi

6.3 Suchbaumverfahren

6.4 Heuristiken

6.5 TD-Q-Lernen

In dem Kapitel 3 Einführung in verstärkendes Lernen, speziell in den Abschnitte 3.6 Temporale Differenz Lernen und **??** Q-Lernen wurde bereits erklärt, wie das TD-Lernen und das Q-Lernen zusammenwirken.

Fassen wir diese beiden Abschnitte kurz zusammen: Temporale Differenz Lernen (TD-Lernen) passt die Nutzen der beobachteten Zustände an die beobachteten Übergänge an. Die Aktualisierungsregel (Gleichung 3.7) des TD-Lernens verwendet die Differenz der Nutzen aufeinanderfolgender Zustände $U\pi(s') - U\pi(s)$, daher die Bezeichnung Temporale Differenz Lernen. Eine alternative TD-Methode ist das Q-Lernen, dass statt Nutzen eine Aktion/Nutzen Repräsentation lernt. Mit der Notation Q(s,a) bezeichnen wir den Wert der Ausführung von Aktion a im Zustand s. Gleichung 3.8 zeigt wie Q-Werte direkt mit Nutzenwerten verknüpft sind.

Der in Abbildung 6.1 skizzierte Algorithmus ist, leicht abgewandelt, im Lehrbuch für künstliche Intelligenz von Russell und Norvig [RN12, S. 974] zu finden.

Dieser ist bereits leicht modifiziert in seiner Notation und nachdem wir die Einzelheiten des Algorithmus geklärt haben, wird die Notation weiter angepasst, bis sie dem Quellcode des Prototypen entspricht.

```
1  def Q–Lernen(s', r', α, γ):
2    if istTerminalzustand(s):
3      Q[s, None] ← r'
4    if s ist nicht None:
5      inkrementiere N_sa[s, a]
6      Q[s, a] ← Q[s, a] + α(N_sa{s, a])
                 * (r + γmax_a' Q[s', a'] − Q[s, a])
7    s, a, r ← s', argmax_a', f(Q[s', a'], N_sa), r'
8    return a
```

Abbildung 6.1 TD-Q-Lernen Algorithmus

Der Q-Lernen Algorithmus verwendet einige persistente (d.h. beständige oder dauerhafte) Variablen und Entitäten. Persistent deshalb, weil sie die einzelnen Funktionsaufrufe überdauern:

- **Q** ist eine Tabelle mit Aktionswerden, indiziert nach Zustand und Aktion. Der Aufruf Q[s, a] liefert z.B. einen Aktionswert (Q-Wert) für eine Aktion a in einem Zustand s. Zu beginn des Lernprozesses sind alle Werte dieser Tabelle leer.

- $\mathbf{N}_{sa}$ ist eine Tabelle mit Häufigkeiten für Zustand/Aktions-Paare. Diese ist wie Q anfangs leer. Jedes mal wenn ein Zustand/Aktions-Paar durchlaufen wird, welches bereits durchlaufen wurde, dann wird der Tabelleneintrag N_{sa}[s, a] inkrementiert d.h. um den Wert 1 erhöht.

- **s** ist der vorhergehende Spielzustand, anfangs leer. Berücksichtigen wir den Zeitlichen Aspekt, dann wäre s zu einem Zeitpunkt t geschrieben s_t und ein darauffolgender Spielzustand wäre $s_t + 1$. Der direkt auf s folgende Spielzustand wird auch als s' (s Prime) bezeichnet.

- **a** ist die vorhergehende Aktion, anfangs leer. Wird die Aktion a im Zustand s ausgeführt, dann wird der Zustand s' bzw. s_{t+1} erreicht. Eine Aktion die in s' ausgeführt werden kann bezeichnen wir als a' oder a_{t+1}.

- **r** ist die Belohnung die dem Agenten von der Umgebung zugeteilt wird, anfangs leer, wenn der Agent eine Aktion a in einem Zustand s ausführt. Wir

können eine Funktion r(s, a) definieren. Die Funktion r(s, a) wird für die meisten Spielzustände s ∈ S den Wert 0 liefern. Für Endzustände der jeweiligen Strategiespiele wird die Funktion r(s, a) andere Werte liefern. Ist r die Belohnung dafür Aktion a in Zustand s auszuführen, dann ist r' (r Prime) die Belohnung dafür Nachfolgeaktion a' in Nachfolgezustand s' auszuführen.

Der Q-Lernen Algorithmus bekommt folgende Eingabeparameter übergeben:

- **s'** ist der aktuelle Spielzustand und gleichzusetzen mit der aktuellen Wahrnehmung des Agenten. Wie bereits erklärt ist s' der Nachfolgezustand von s.
- **r'** ist das Belohnungssignal, welches der Agent erhält, wenn er eine Aktion a' im Zustand s' ausführt.
- **α** ist bestimmt über die Lernrate des Algorithmus. Der Wert von α ist in der Regel zwischen 0 und 1. Eine hohe Lernrate (α nahe 1) bedeutet, dass die Aktualisierung des Q-Werts stärker ist. Bei einer niedrigen Lernrate ist die Aktualisierung schwächer. Der Ausdruck $\alpha(N_{sa}[s, a])$ im TD-Q-Lernen Algorithmus bedeutet, aktualisiere Q-Werte für neue noch unbekannte Zustands/ Aktions-Paare Stärker und vertraue den Q-Werten von bereits öfter besuchten Zustand/ Aktions-Paaren, sprich je öfter ein Zustand/Aktions-Paar bereits besucht wurde, umso weniger muss der Q-Werte aktualisiert werden.
- γ ist der Abschwächungsfaktor (eng. discounting factor). Im fachlichen Umfeld des verstärkenden Lernens wird dieser Abschwächungsfaktor bei Modellen mit unendlichen Horizont verwendet. Endet eine Aktionssequenz in einem Markov-Entscheidungsprozess nicht, dann ist diese unendlich. Um Probleme dieser Klasse trotzdem handhaben zu können, wird für die Berechnung des erwarteten Nutzens $U^{\pi}(s)$ (siehe **??** Optimale Taktiken Gleichung für den erwarteten Nutzen 3.1) eines Zustands s der Abschwächungsfaktor verwendet. Da sowohl Tic Tac Toe als auch Reversi, nach einer maximalen Anzahl von Aktionen, immer in einem Endzustand terminieren, werden wir den Abschwächungsfaktor gleich 1 setzen. Ein Abschwächungsfaktor von 1 bedeutet, dass Belohnungen in der Zukunft genau so Wertvoll sind wie unmittelbare Belohnungen.

Konzentrieren wir uns nachfolgend auf die Aktualisierung dieser Q-Werte.

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{6.1}$$

Kapitel 7

Validierung

In diesem Kapitel: //TODO Einführung in das Kapitel

7.1 Logiktest der Strategiespiele Tic Tac Toe und Reversi

7.2 Agententest

Empirisches Protokoll

7.2.1 Bewertungskriterien

7.2.2 Persistenz der Agentenerfahrung

Kapitel 8

Auswertung

In diesem Kapitel: //TODO Einführung in das Kapitel

8.1 Konvergenz des TD-Q-Lernens

8.1.1 Generalisierung oder Funktionsannäherung

Samuels Dame-Spiel Arthur L. Samuel schrieb 1955 ein Programm, dass Dame spielen konnte und mit einem einfachen Lernverfahren seine Parameter verbessern konnte. Sein Programm hatte dabei jedoch Zugriff auf eine große Zahl von archivierten Spielen, bei denen jeder einzelne Zug von Experten bewertet war (Überwachtes Lernen zur Unterstützung des verstärkenden Lernens). Damit verbesserte das Programm seine Bewertungsfunktion. Um eine noch weitere Verbesserung zu erreichen, ließ Samuel dein Programm gegen sich selbst spielen. Das Credit Assignment löste er auf einfache Weise. Für jede einzelne Stellung während eines Spiels vergleicht er die Bewertung durch die Funktion B(s) mit der durch Alpha-Beta-Pruning berechneten Bewertung und verändert B(s) entsprechend. 1961 besiegte sein Dame-Programm den viertbesten Damespieler der USA. Mit dieser bahnbrechenden Arbeit war Samuel seiner Zeit um fast dreißig Jahre voraus [Ert16, S. 120 f.].

Agent mit TD-Lernen Die Aufgabe des Agenten mit TD-Lernen ist, dass verbessern einer gegebenen Heuristik. Unter Verwendung dieser möglicherweise verbesserten Heuristik, soll der Agent eine möglichst optimale Aktion auswählen. Der Agent verbessert die Bewertungsfunktion durch Aktualisierung bzw. Anpassung der Parameter $\theta = \theta_1, ...\theta_n$.

$$\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + ... + \theta_n f_n(s)$$

8.1.2 Neuronales Lernen

TD-Gammon Das TD-Lernen zusammen mit einem Backpropagation-Netz mit 40 bis 80 verdeckten Neuronen wurde sehr erfolgreich angewendet in TD-Gammon, einem Programm zum Spielen von Backgammon, programmiert vom Entwickler Gerald Tesauro im Jahr 1992. Die einzige direkte Belohnung für das Programm ist das Ergebnis am Ende eines Spiels. Eine optimierte Version des Programms mit einer 2-Züge-Vorausschau wurde mit 1,5 Millionen Spielen gegen sich selbst trainiert. Es besiegte damit Weltklassespieler und spielt so gut wie die drei besten menschlichen Spieler [Ert16, S. 304].

Sind Lernverfahren überhaupt Sinnvoll?

8.2 Gegenüberstellung der Lernverfahren

Belastbarkeit und Grenzen der Lernverfahren ?
Optimale Anwendungsspiele für die Lernverfahren?
Bewertung der Strategien?

Überwachtes Lernen

Wert-Iteration und dynamische Programmierung

TD-Lernen

Q-Lernen

Funktionsannäherung

8.3 Gegenüberstellung überwachtes und verstärkendes Lernen

Ein überwachtes Lernverfahren wird zuerst mittels eines Trainingssets kontrolliert belehrt. Das Trainingsset für ein überwachtes Lernverfahren besteht aus verschiedenen Eigenschaften (Features) und einer den Eigenschaften zugeordneten Klasse beziehungsweise Zielvariable (Target variable). Die Qualität der Zielwerte kann mittels Testsets ermittelt werden. Ein Testset ist ein Datenset bestehend aus Eigenschaften ohne die dazugehörigen Klassen. Abbildung 8.1 zeigt, wie ein Trainingsset aussehen könnte[Har12, S. 8]. Die Spalten Gewicht, Flügelspanne, Schwimmhäute

und Rückenfarbe sind die Eigenschaften und die Spalte Spezies beinhaltet die Zielvariablen. Das überwachte Lernverfahren lernt mittels des Trainingsdatensets die Beziehungen zwischen den Eigenschaften und der Klassen.

	Gewicht (g)	Flügelspanne (cm)	Füße mit Schwimmhäuten?	Rückenfarbe	Spezies
1	1000,1	125,0	Nein	Braun	Buteo jamaicensis
2	3000,7	200,0	Nein	Grau	Sagittarius serpentarius
3	3300,0	220,3	Nein	Grau	Sagittarius serpentarius
4	4100,0	136,0	Ja	Schwarz	Gavia immer
5	3,0	11,0	Nein	Grün	Calothorax lucifer
6	570,0	75,0	Nein	Schwarz	Campephilus principalis

Abbildung 8.1 Vogelspezies Klassifikation basierend auf vier Eigenschaften

Bezogen auf ein Testdatenset wäre eine Aktion des Agenten, dass nennen einer Klasse hinsichtlich der gelernten Zusammenhänge aus den Trainingsdaten. Teilt man die Anzahl der korrekten Vorschläge durch die Anzahl aller Versuche, dann erhält man eine Kennzahl für die Qualität der Vorhersage. Sind alle Klassen der Testinstanzen richtig vorhergesagt, dann ist die Kennzahl genau Eins. Jede Zeile eines Testsets und jede Zeile eines Trainingssets ohne die Zielvariablen ist eine Instanz.

Im Gegensatz zu den Lernverfahren beim überwachten Lernen fehlt dem Agenten beim verstärkenden Lernen ein Lehrer (z.B. das Trainingsset), der dem Agenten genau sagt ob seine Aktionen richtig oder falsch sind. Den Experten für Strategiespiele wie Schach fällt es außerdem sehr schwer für jede mögliche Stellung eine angemessen ausformulierte Bewertung bereitzustellen. Zudem ist die Spielzustandsmenge, also die Anzahl an möglichen Stellungen, beim Schach sehr hoch (siehe Abschnitt 2.3.1 Minimax). Selbst wenn es den Experten leicht fallen würde, jede Stellungen explizit bewerten zu können, dann ist die Anzahl der Stellungen immer noch so groß, dass diese Art der Lehre extrem kostspielig wäre.

Problembereiche des verstärkenden Lernens sind:

- Lernen von Problemen mit sehr großen Zustand- und Aktionsmengen
- Lernen von unbekannten oder sehr komplexen Probleme, für die keine Erfahrungswerte (Testsets) existieren oder diese zu komplex zu beschreiben sind

Verstärkendes Lernen kann trotzdem vom überwachten Lernen profitieren, denn es ist z.B. möglich den Agenten in der Anfangsphase des verstärkten Lernens explizit zu programmieren und ihn dadurch auf bestimmte Auffälligkeiten oder Muster aufmerksam zu machen. Ist es zu kompliziert dies explizit zu programmieren, dann

kann auch ein Mensch dem Agenten die richtigen Aktionen vorgeben [Ert16, S. 306].

Sehr nützlich werden diese beiden Unterstützungen der Anfangsphase des verstärkenden Lernens, sobald die Dimensionen der Umwelt oder des Agenten eine bestimmte Größe überschreiten. Eine Aktionsdimension kann sehr wenige Aktionen beinhalten zum Beispiel die vier Aktionen bewege dich nach oben, unten, rechts oder links. Roboter die dem Menschen nachempfunden sind verfügen über bis zu 50 verschiedene Motoren für die einzelnen Gelenke. Diese müssen gleichzeitig angesteuert werden, was zu einem 50-dimensionalen Zustandsraum und einem 50-dimensionalen Aktionenraum führt[Ert16, vgl. 305 f.]. Bei solch großen Dimensionen können die Laufzeiten einiger verstärkender Lernverfahren massiv ansteigen, bis sie praktisch nicht mehr handhabbar sind. Gerade in der Anfangsphase des verstärkten Lernens kann darum ein Eingriff mittels überwachtem Lernen sehr Laufzeit schonend sein.

Zusammengefasst kann das überwachte Lernen die Problematiken des verstärkenden Lernens nicht alleine lösen, aber überwachtes Lernen kann das verstärkende Lernen in bestimmten Lernphasen unterstützen und die Leistung des verstärkenden Lernens verbessern. Später in dieser Arbeit werden wir eine Anwendung betrachten in der unter anderem überwachtes Lernen eingesetzt wurde, um einen verstärkenden Lernalgorithmus zu unterstützen (Samuels Dame-Spiel).

Literatur

[Alp08] Ethem Alpaydin. *Maschinelles Lernen*. 1. Aufl. Oldenbourg, 2008.

[Bei14] Christoph Beierle. *Methoden wissensbasierter Systeme: Grundlagen, Algorithmen, Anwendungen*. 5. Aufl. Springer, 2014.

[Ert16] Wolfgang Ertel. *Grundkurs Künstliche Intelligenz: Eine prakmatische Einführung*. 4. Aufl. Springer, 2016.

[Har12] Peter Harrington. *Machine Learning: IN ACTION*. 1. Aufl. Manning, 2012.

[KLM96] Leslie Pack Kaelblig, Michael L. Littman und Andrew W. Moore. „Reinforcement Learning: A Survey". In: *Jornal of Artificial Intelligence Research* 4 (1996), S. 237–285.

[Lö93] Jan Löschner. *Künstliche Intelligenz: Ein Handwörterbuch für Ingenieure*. 1. Aufl. VDI, 1993.

[Ras16] Sebastian Raschka. *Machine Learning mit Python*. 1. Aufl. MIT Press, 2016.

[RN12] Stuart Russel und Peter Norvig. *Künstliche Intelligenz: Ein moderner Ansatz*. 3. Aufl. Pearson, 2012.

[Zob70] Albert L. Zobrist. „A new hashing method with application for game playing". In: *Technical Report* 88 (1970), S. 1–12.