Entwicklung eines TD(λ)-basierten Skip-Bo-Spielers

Studienarbeit
von
Sanja Jahnke

Albert-Ludwigs-Universität Freiburg
Fakultät für angewandte Wissenschaften
Institut für Informatik
Professor Nebel

Februar 2007

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Datum Unterschrift

Danksagung

An dieser Stelle bedanke ich mich bei Herrn Prof. Nebel, der diese Studienarbeit möglich gemacht hat, und bei Herrn Dipl. Inf. Sebastian Kupferschmid für die Unterstützung und die aufgebrachte Zeit. Er stand mir bei Fragen jederzeit hilfreich zur Seite.

Zusammenfassung

Diese Studienarbeit beschäftigt sich mit Skip-Bo, einem Kartenspiel für zwei bis sechs Spieler, und der Implementierung verschiedener Skip-Bo-Agenten. Es werden regelbasierte Agenten entwickelt, und der Temporal Differences- Algorithmus zur Erstellung selbstlernender Agenten verwendet. Bei dieser Methode werden die Parameter einer Bewertungsfunktion so angepasst, dass der erwartete langfristige Nutzen eines Spielzustandes möglichst genau vorhergesagt wird. Die so entstandenen Agenten und ihre Spielfähigkeiten können unter http://www.td-skip-bo.de.vu getestet werden.

Inhaltsverzeichnis

1 Einführung 4
2 Die Skip-Bo-Spielregeln 4
2.1 Das Spielfeld 4
2.2 Der Spielablauf 5
2.3 Kniffe und Tricks 6
3 Die regelbasierten Skip-Bo-Spieler 8
3.1 Basic Agent A^B 8
3.2 Mature Agent A^M 10
4 Temporal-Difference-Lernen 10
4.1 Der TD-Ansatz 11
4.2 Reinforcement-Lernen 12
4.3 Der TD(λ)-Algorithmus 13
4.4 Die lernenden Skip-Bo-Spieler: Anwendung des TD-Lernens 17
4.4.1 TD lernender Agent A^{TD} 17
4.4.2 Learning Agent A^{Lea} - Bewertung der Aktion 19
5 Evaluation 23
5.1 A^Bs Spielverhalten 24
5.2 A^Ms Spielverhalten 24
5.3 A^{TD}s Spielverhalten 25
5.4 A^{Lea}s Spielverhalten 25
5.5 Gemischte Runde 26
6 Das Programm – Skip-Bo online 26
7 Zusammenfassung und Ausblick 29
8 Quellenverzeichnis 30

1 Einführung

Skip-Bo ist ein Kartenspiel. Die Skip-Bo-Karten haben Werte von Eins bis Zwölf und werden durch Joker ergänzt. Aus ihnen werden während des Spiels verschiedene Kartenstapel aufgebaut, die teilweise auch wieder abgebaut werden können. Ziel des Spiels ist es der Erste zu sein, der seinen Vorratsstapel geleert hat. Jeder Spieler besitzt einen solchen Vorratsstapel, auf dem zu Beginn des Spiels 15 Karten liegen, von denen jeweils nur die Oberste sichtbar ist.
Die Informationen des Spiels sind unvollständig, da man nicht weiß welche Karten man als nächstes erhält, welche Karten der Gegner auf der Hand hat und welche unter der obersten Karte des Vorratsstapels liegen. Deshalb muss man, um trotzdem gewinnen zu können, geschickt planen und die vorhandenen Ressourcen – die erhaltenen Karten und die Spielstapel – intelligent nutzen.
Dieses Spiel eignet sich aus verschiedenen Gründen zur Entwicklung von Spiel-Agenten und insbesondere auch zum Testen der Anwendung des TD-Algorithmus. Zum einen kann die Spielumgebung und die aktuelle Spielsituation gut dargestellt werden, und die erlaubten Spielzüge sind von den verbotenen eindeutig zu unterscheiden und einfach umzusetzen. Zum anderen muss ein Spieler planvoll intelligente Spielzüge machen, um erfolgreich spielen zu können. Das heißt, dass die erreichte Intelligenz und somit das Können eines Agenten anhand seines Spielerfolges messbar ist.
Der TD-Algorithmus ist eine Methode, mit der ein Spieler im Spiel gegen sich selbst, ohne jegliche Vorkenntnisse und ohne menschlichen Lehrer lernen kann, welche Spielzüge Erfolg versprechen und zum Gewinn führen. Entwickelt wurde diese Lerntechnik von Samuel [6]. Die weiter entwickelte Form TD(λ) von Sutton [4] wird heute zumeist auf Brettspiele wie Schach oder Backgammon angewendet. Der bislang größte Erfolg des TD(λ)-Algorithmus ist Tesauros TD-Gammon [2], dessen Backgammon spielender Agent alle bisherigen Agenten übertraf und sich auch gegen menschliche Profispieler bewies.
Diese Arbeit wird zunächst in Abschnitt 2 die Spielregen und die spezifischen Kniffe des Skip-Bo-Spiels erläutern und anschließend in Abschnitt 3 zwei regelbasierten Spieler beschreiben. Der TD- Algorithmus und seine Anwendung bei Skip-Bo werden in Abschnitt 4 ausgeführt. In Abschnitt 5 werden die Spielweisen und -ergebnisse der verschiedenen Skip-Bo-Agenten verglichen. Abschnitt 6 enthält zusammenfassende, abschließende Bemerkungen.

2 Die Skip-Bo-Spielregeln

Skip-Bo ist ein Kartenspiel für zwei bis sechs Spieler. Gespielt wird hierbei mit insgesamt 162 Karten, von denen 144 die Werte Eins bis Zwölf haben, und die restlichen 18 Joker sind.

2.1 Das Spielfeld

Die Skip-Bo-Spielkarten, zu sehen in Abbildung 2.1, haben Werte von Eins bis Zwölf, aber keine verschiedenen Farben, wie es sie beispielsweise beim Poker gibt. Der Joker des Spiels heißt – wie das Spiel selbst – Skip-Bo, und kann anstelle jeder Zahl gespielt werden, d.h. jeden möglichen Wert annehmen.

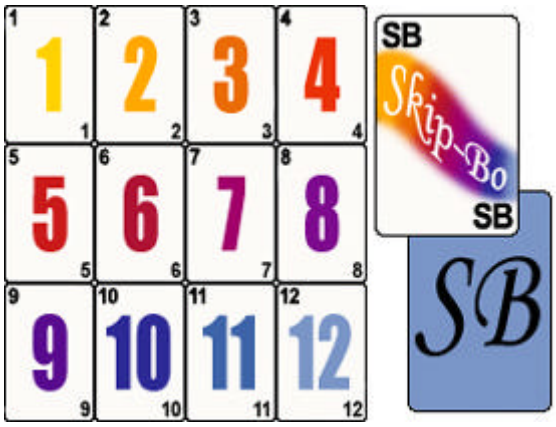

Abb. 2.1: Das hier verwendete Kartenset: links: die Kartenwerte Eins bis Zwölf, rechts oben: der Skip-Bo-Joker, rechts unten: die Karten-Rückseite.

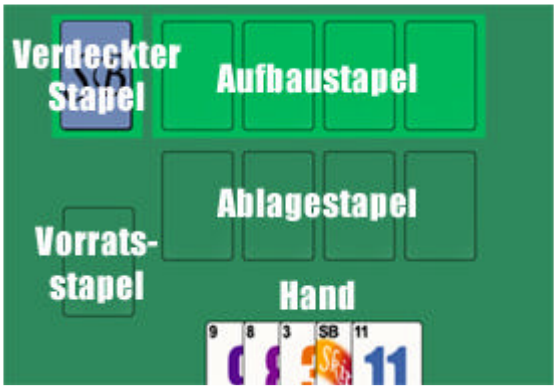

Abb. 2.2: Die verschiedenen Spielstapel für einen Spieler. Die hellgrün unterlegten Stapel sind für alle zugänglich. Die anderen gehören nur dem Spieler selbst.

Abbildung 2.2 zeigt ausschnittsweise das Spielfeld für einen Spieler, wobei die hellgrün unterlegten Bereiche für alle Spieler zugänglich sind. Das Spielfeld besteht aus vier verschiedenen Arten von Kartenstapeln:

1. **Der Verdeckte Stapel:**
   Von hier werden zusätzliche Karten gezogen; alle Karten, die nicht im Umlauf sind, befinden sich auf diesem Stapel.

2. **Die Aufbaustapel:**
   Vier Aufbaustapel bilden das Zentrum des Spielfelds. Sie sind zu Beginn des Spiels leer und werden von allen Spielern verwendet: Auf einen leeren Stapel kann eine Eins oder ein Skip-Bo gelegt werden, und auf einen angefangenen Stapel die jeweils nächst höhere Karte. Sobald auf einem Aufbaustapel 12 Karten liegen wird er geleert, und ein neuer Stapel kann an seiner Stelle angefangen werden.

3. **Die Ablagestapel:**
   Jeder Spieler hat vor sich vier Ablagestapel, die nur er benutzten darf. Auch sie sind anfangs leer, können später aber beliebig viele Karten in beliebiger Reiheinfolge enthalten. Diese Karten werden während des Spiels vom jeweiligen Spieler hinzugefügt und können – von oben weg – wieder entfernt und ausgespielt werden.

4. **Die Vorratsstapel:**
   Jeder Spieler hat einen Vorratsstapel, der zu Beginn 15 Karten enthält und so schnell wie möglich geleert werden muss.

Zusätzlich zu den vorgestellten Stapeln hält jeder Spieler – für die anderen verdeckt – bis zu fünf Karten auf der Hand.

2.2 Der Spielablauf

Ziel des Spiels ist es alle Karten des Vorratsstapels loszuwerden; derjenige, der dies als Erster schafft, gewinnt.
Das Spiel beginnt, indem Spieler 1 fünf Karten vom Verdeckten Stapel zieht. Er darf dann so viele Karten von seinem Vorratsstapel und seiner Hand ausspielen, wie er kann

und will. Wenn er alle fünf Karten seiner Hand ausgespielt hat, bekommt er fünf neue und spielt weiter. Seinen Spielzug beendet er durch das Abspielen einer seiner Handkarten auf einen seiner Ablagestapel. Dann ist der nächste Spieler an der Reihe.
Bei jeder weiteren Runde füllt ein Spieler zuerst seine Hand mit Karten aus dem verdeckten Stapel auf, so dass sie wieder fünf davon enthält. Anschließend kann er auf die Aufbaustapel weitere Karten legen, indem er die oberste Karte des Vorratsstapels, der Ablagestapel oder eine Karte aus seiner Hand spielt.

2.3 Kniffe und Tricks

Bei Skip-Bo gibt es für viele Situationen Daumenregeln für die bestmögliche Aktion. Man weiß zwar nie, welche Karten man von dem Verdeckten Stapel erhält, welche Karte unter der obersten Karte des Vorratsstapels liegt oder welche Karten der Gegner ausspielen wird, aber die Reihenfolge, in der man seine Karten ausspielt, welche Karten man auf der Hand behält und der Inhalt der eigenen Ablagestapel sind kontrollierbar. Beispielsweise ist es immer am besten die Karten des Vorratsstapels sobald wie möglich auszuspielen, da derjenige gewinnt, der diesen Stapel als erstes geleert hat. Die wichtigsten Daumenregeln lassen sich durch die folgenden Kniffe beschreiben.

Kniff 1 (*Vorratsstapel favorisieren*):
Die einfachste Regel ist immer den eigenen Vorratsstapel im Auge zu behalten, und, falls möglich, immer zuerst die dort liegende Karte auszuspielen, auch wenn man auf der Hand oder auf einem Ablagestapel eine Karte mit gleichem Wert hat. Die dahinter stehende Idee ist – wie schon oben erwähnt – der Erste zu sein, der seinen Vorratsstapel geleert hat. Abbildung 2.3 zeigt einen solchen Fall.

Kniff 2.1 (*Rangfolge der Aufbaustapel*):
Diese Regel leitet sich direkt aus der Vorhergehenden ab: Man sollte immer zuerst nur die Karten legen, die helfen die Karte des Vorratsstapels zu spielen, und die anderen erst danach. Grund hierfür ist, dass man nicht weiß, welche Karte nach dem Spielen der obersten Vorratsstapel-Karte als nächstes auf dem Vorratsstapel liegt. Wenn man nun aber Karten ausspielt, die nicht direkt fürs Spielen der Vorratsstapel-Karte benötigte werden, kann es geschehen, dass man nun die nachfolgende Karte des Vorratsstapels nicht mehr legen kann, da diese den zuvor gespielten Wert hat. Abbildung 2.4 verdeutlicht diesen Fall anhand eines Beispiels.

Kniff 2.2 (*Rangfolge von Hand und Ablagestapel*):
Ebenso gilt es zu überlegen, ob man, wenn auf der Hand und auf einem Ablagestapel jeweils Karten mit dem gleichen, spielbaren Wert liegen, besser die Karte vom Ablagestapel oder die aus der Hand nimmt. Wenn man die Karte aus der Hand nimmt, hat das den Vorteil, dass man bei der nächsten Runde mehr neue Karten erhält. Dahingegen kann unter der obersten Karte des Ablagestapels eine weitere liegen, die man benötigt, oder die sonst zumindest über die Zeit immer tiefer in den Stapel rutscht, so dass sie – wenn sie benötigt wird – nicht mehr erreichbar ist. Für einen Spieler ist also sowohl die Rangfolge „zuerst von der Hand, dann vom Ablagestapel“ als auch „zuerst vom Ablagestapel, dann von der Hand“ oder eine Kombination davon denkbar. Da der Kniff ist situationsabhängig ist, sollte in ca. 65% der Fälle der Ablagestapel vorgezogen werden.

Kniff 3 (*Geschicktes Ablegen*):
Als drittes sollten die Ablagestapel sinnvoll genutzt werden. Die Karten sollten möglichst so auf den Ablagestapeln platziert werden, dass sie – wenn benötigt – wieder

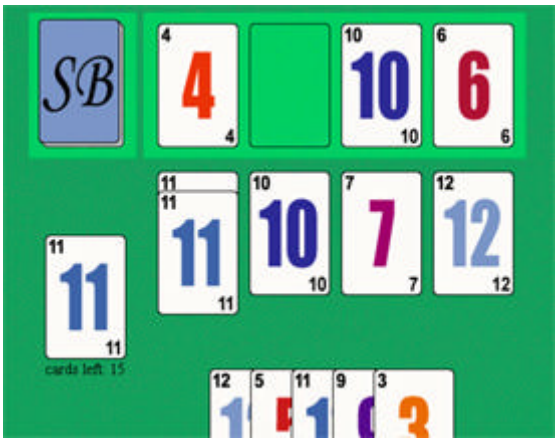

Abb. 2.3: Die Karte des Stock Piles (hier eine Elf) sollte immer zuerst gelegt werden. In diesem beispielhaf-ten Fall besteht gleichzeitig die Mög-lichkeit eine Elf aus der Hand oder vom Ablagestapel zu spielen, was jedoch dem Ziel zu gewinnen weit weniger zuträglich ist.

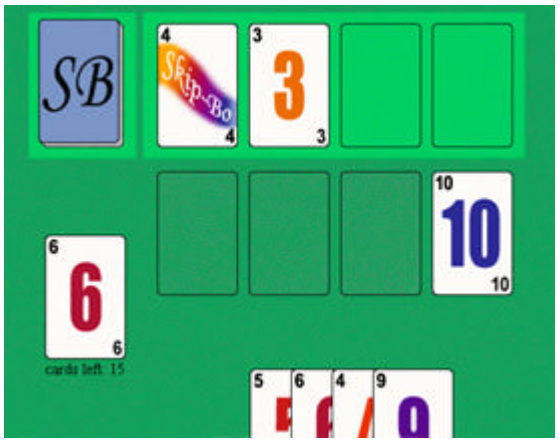

Abb. 2.4: Hier können die Karten Vier und Fünf aus der Hand zugleich gespielt werden. Spielt man jedoch die Vier vor der Fünf und der Sechs vom Vorrats-stapel, kann es sein, dass unter der Sechs eine Vier liegt (mit 7,4%iger Wahrscheinlichkeit), die nun nicht mehr gespielt werden kann.

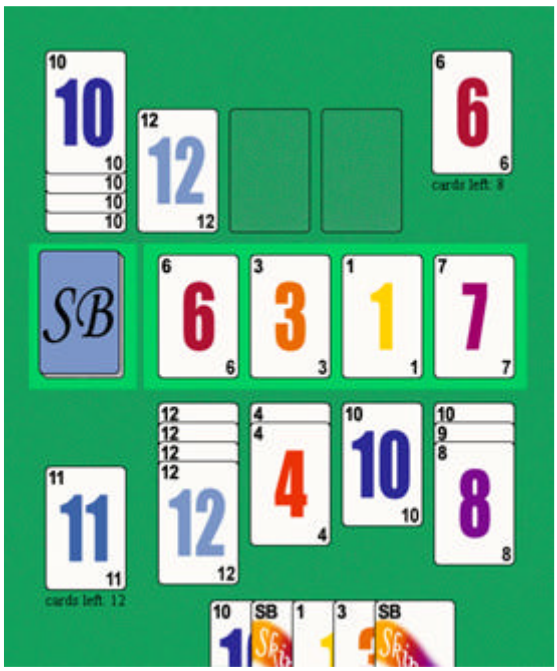

Abb. 2.5: Diese Sicht über das Spielfeld für zwei Spieler zeigt die Ablage-strategien: jeweils gleiche Werte auf-einander (wie bei dem linken gegne-rischen und den eigenen linken Ablage-stapeln) und „Straße" (auf dem rechten eigenen Ablagestapel), die nun ermög-licht die oberste Karte des Vorrats-stapels zu spielen, indem zuerst die Acht, Neun und Zehn auf dem vierten Aufbaustapel platziert werden.

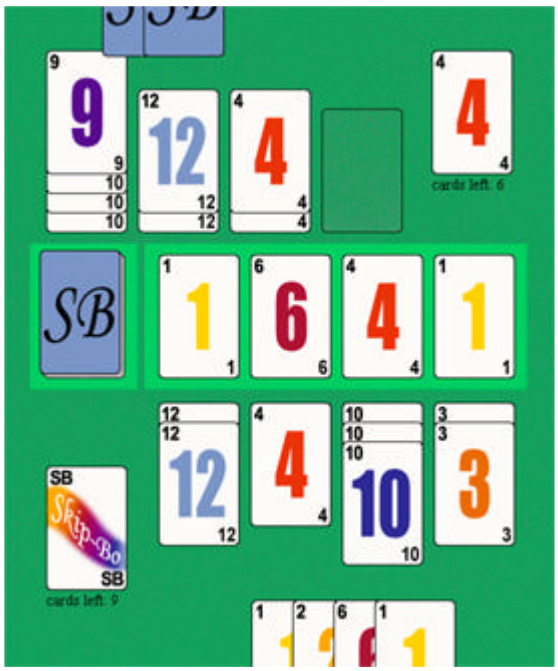

Abb. 2.6: Hier sollte die Zwei aus der Hand und die Drei vom vierten Ablage-stapel nur gespielt werden, wenn auch die Vier vom zweiten Ablagestapel gelegt wird, denn diesen Wert hat die gegnerische Vorratsstapel-Karte. Davor sollte allerdings der Skip-Bo vom eige-nen Vorratsstapel gelegt werden, und zwar entweder auf die Vier oder die Sechs, da die Zwei auf der Hand liegt, aber keine Fünf oder Sieben spielbar ist. Würde man sie auf eine Eins legen und dann keine Drei und Vier darauf legen, wäre das hier zum Vorteil des Gegners.

erreichbar sind. Dies wird dadurch erschwert, dass man die Karten jeweils nur oben auf einen der Stapel legen darf, und auch nur die oberste Karte wieder entnehmen. Eine mögliche Lösung dafür ist es gleichwertige Karten aufeinander zu legen. Alternativ dazu kann man auch immer eine niedrigere auf eine höhere Karte legen, um so eine „Straße“ zu bilden, die rückwärts wieder ausgespielt werden kann. Z.B. kann eine auf einem Ablagestapel (von unten nach oben) angesammelte Straße 4, 3, 2, 1 auf einen leeren Aufbaustapel ausgespielt werden. Abbildung 2.5 zeigt diese beiden Methoden.

Kniff 4 (*Gebrauch der Skip-Bos*):
Der vierte Kniffbetrifft die Joker Karten. Diese sollten nur in sinnvollen Situationen gespielt werden, denn ansonsten fehlt der Skip-Bo, wenn man ihn später wirklich braucht. Sinnvoll ist es beispielsweise einen fehlenden Kartenwert zu ersetzen, um danach die Karte vom Vorratsstapel legen zu können. Verschwenderisch wäre es, einen Skip-Bo anstelle einer Karte, die man ebenfalls auf der Hand hält, zu spielen. Auch das Ausspielen von Skip-Bos vom Vorratsstapel sollte nicht einfach wahllos geschehen, denn die Chance die darunter folgende Karte spielen zu können, sollte möglichst maximiert werden. Abbildung 2.6 verdeutlicht dies.

Kniff 5 (*Beobachten des Gegners*):
Als letztes gilt es nun noch die Kartenwerte auf Vorrats- und Ablagestapel der Gegenspieler im Auge zu behalten, um nicht zu ermöglichen (bzw. zu verhindern), dass diese die Karten von ihren Vorratsstapeln ausspielen können. Eine solche Situation wird in Abbildung 2.6 dargestellt.

Mithilfe dieser Kniffe können nun regelbasierte Skip-Bo-Agenten entwickelt werden.

3 Die regelbasierten Skip-Bo-Spieler

Dieser Abschnitt behandelt die zwei regelbasierten Skip-Bo-Spieler Basic und Mature Agent. Diese sollen die Kniffe aus Abschnitt 2.3 durch feste, von mir vorgegebene Regeln umsetzen. Dabei benutzt Basic nur die wichtigsten Kniffe – 1: Vorratsstapel favorisieren, 2.2: Rangfolge (zuerst Ablagestapel, dann Hand), 3: Gebrauch der Ablagestapel und 4: Gebrauch der Skip-Bos –, und Mature alle sechs. Anschließend wird im beschrieben, wie Agenten durch Anpassung von Parametern mit dem TD-Algorithmus lernen Skip-Bo zu spielen.

3.1 Basic Agent A^B

Basic Agent A^B ist ein eher einfacher Spieler, der eigentlich nur die Grundregeln des Spiels beherrscht. Er wurde hauptsächlich zu Vergleichszwecken programmiert und spielt ungefähr auf dem Level eines wenig vorausschauenden Anfängers.
Die grundlegende Routine eines Spielzuges von A^B ist wie folgt:
Zuerst wird getestet, ob die oberste Karte des Vorratsstapels gelegt werden kann. Falls ja, wird dies wiederholt, falls nein werden die anderen Karten betrachtet, und zwar beginnend mit den obersten Karten der Ablagestapel und danach die Karten auf der Hand. Wenn eine dieser Karten gespielt werden kann, wird der entsprechende Spielzug gemacht, und dann sofort wieder die Karte des Vorratsstapels auf Spielbarkeit geprüft. Damit wird sichergestellt, dass die Karte des Vorratsstapels bei der ersten sich bietenden Möglichkeit gespielt wird, und nicht etwa eine gleichwertige Karte aus der Hand. Sobald keine Karte mehr gespielt werden kann folgt die Ablage-Aktion, die weiter unten beschrieben wird. Abbildung 3.1 fasst dies in Pseudocode zusammen.

```
public void makeYourMove() {
  while (cards are playable) {
    while (card from stock[1] pile is playable) {
      play card from stock pile to a building[2] pile
    }
    try to play a card from discard[3] pile to any building pile
    if (no card is played from discard pile) {
      try to play a card from hand to any building pile
    }
  }
  discard()[4]
}
```

Abb. 3.1: Grundlegende Routine von A^B.

1: Vorratsstapel; 2: Aufbaustapel; 3: Ablagestapel; 4: Ablage-Aktion.

Skip-Bos aus der Hand werden nur auf den ***„besten Aufbaustapel“*** gespielt, und das auch nur, wenn die darauf folgende Karte auf A^Bs Hand oder auf den Ablagestapeln vorhanden ist, oder wenn sich auf ihrer Hand nur noch Skip-Bos befinden. Als ***bester Aufbaustapel*** wird der Aufbaustapel bezeichnet, auf den man am wenigsten Karten legen muss, so dass die oberste Karte des Vorratsstapels spielbar ist.
Skip-Bos vom Vorratsstapel werden auf einen zufälligen Aufbaustapel gelegt. Ausnahmen dafür bilden die Fälle, in welchen mehrere Aufbaustapel den gleichen Wert haben. In diesen wird der Skip-Bo auf einen davon gelegt.
Bei Ablage-Aktion wird zuerst getestet, ob eine Karte auf der Hand den gleichen Wert hat, wie die auf einem der Ablagestapel, und falls ja, wird diese dort abgelegt. Ansonsten wird geprüft, ob eine Karte als „Straße“ (siehe Kniff 3) abgelegt werden kann, und falls auch das nicht geht, wird die am häufigsten auftretende Karte mit dem höchsten Wert ausgewählt und – falls möglich – auf einen leeren Ablagestapel gelegt. Abbildung 3.2 fasst dies nochmals in Pseudocode zusammen.

```
public void discard() {
  try to discard equal valued card
  if (not card is played to discard pile yet) {
    try to discard ‘street’
  }
  if (not card is played to discard pile yet) {
    for (i = 1; i <= 12; i++) {
      x_i = how many cards with value i are on hand
    }
    y = highest x_i
    if (only for one i = 1,…,12 is x_i = y) {
      discard one of the cards with value i
    }
    else {
      select i with x_i = y and i > j for all j: x_j = y
      discard one of the cards with value i
    }
  }
}
```

Abb. 3.2: Ablage-Aktion.

A^B ist also ein relativ einfacher Spieler, jedoch nicht der der schlechteste. Ersetzt man beispielsweise die Ablagestrategie durch zufälliges Ablegen, dann spielt der dadurch entstehende Agent A^{B-} gleich um einiges schlechter (siehe Abschnitt 5). Das zeigt, wie wichtig eine gute Ablagestrategie ist, und wie sehr die Nicht-Nutzung eines Kniffes die Gewinnchancen beeinträchtigt.

3.2 Mature Agent A^M

Der Mature Agent A^M ist eine Erweiterung von A^B und baut auf dessen Spielweise auf. Die grundlegende Routine eines Spielzuges enthält einen zusätzlichen Schritt: direkt nach der Überprüfung des Stock Piles wird getestet, ob eine Karte von einem Ablagestapel oder aus der Hand auf den besten Aufbaustapel gelegt werden kann (siehe Kniff 2.1). Erst wenn das nicht mehr möglich ist, werden die anderen Karten auf Spielbarkeit überprüft. Die geänderte Grundroutine ist in Abbildung 3.3 dargestellt.
Zusätzlich wird bei jedem Ausspielen einer Karte aus der Hand oder von den Ablagestapeln getestet, ob dies dem Gegner einen Vorteil vermittelt (siehe Kniff 5). D.h., wenn eine Karte gelegt werden soll, durch die der Gegner in die Lage kommt seine Vorratsstapel-Karte zu spielen (und ohne die Karte nicht, oder zumindest nicht nach Wissen von A^M, der ja die Hand des Gegners nicht einsehen kann), und A^M selbst keine weitere(n) Karte(n) legen kann, um dies zu verhindern, wird diese Karte nicht gespielt, sondern behalten. Wenn dieser Fall bei einer vom Vorratsstapel zu spielenden Skip-Bo Karte eintritt, wird die Karte einfach auf einen der anderen Aufbaustapel gelegt. Alles andere – auch die Ablage-Aktion – ist wie bei A^B.

```
public void makeYourMove() {
  while (cards are playable) {
    while (card from stock pile is playable) {
      play card from stock pile to a building pile
    }
    try to play a card from discard pile or hand to best building
    pile
    if (no card is played yet*) {
      try to play a card from discard pile to a building pile
    }
    if (no card is played yet*) {
      try to play a card from hand to a building pile
    }
  }
  discard()
}
```

Abb. 3.3: Grundlegende Routine des A^M. Bei jedem Spielen einer Karte aus der Hand oder von den Ablagestapeln wird zusätzlich getestet, ob dies (ausschließlich) dem Gegner einen Vorteil vermittelt. Der weitere Unterschied zu A^B ist fett dargestellt.

(*) apart from the stock pile's cards.

Die weiteren, lernenden Agenten basieren auf dem TD-Algorithmus. Sie trainieren ihre Fähigkeiten im Spiel gegen sich selbst und gegen andere Spieler. Dies geschieht durch Lernen der Gewichtung verschiedener Parameter, mit denen der Nutzen eines Spielzustandes angegeben werden kann. Die Trainingspartner sollten dabei jedoch nicht wesentlich besser sein als die Lernenden. Die genaue Funktionsweise und Anwendung dieses Algorithmus und die dadurch entstandenen Spielfähigkeiten der Agenten werden in den folgenden Abschnitten erläutert.

4 Temporal-Difference-Lernen

Dieser Abschnitt beschreibt zuerst allgemein den TD-Ansatz. Anschließend wird das Reinforcement-Lernen, auf dem der TD- Algorithmus basiert, erklärt. Und zuletzt wird der TD(λ)-Algorithmus selbst beschrieben, sowie die Experimente, welche zum Skip-

Bo-Lernen gemacht wurden. Die Begriffe „Temporal Difference“ und „Reinforcement“ werden der Klarheit wegen verwendet, da es für sie keine bekannten deutschen Äquivalente gibt.

4.1 Der TD-Ansatz

A. Samuel entwickelte den TD-Ansatz in Anwendung für ein Schachspiel [6]. Später wurde er von Sutton [4] zu TD(λ) weiterentwickelt. TD(λ) ist ein Algorithmus, der den erwarteten, langfristigen Nutzen eines stochastischen, dynamischen Systems approximiert. Dafür wird eine mehrwertige Funktion verwendet, welche einem Zustand seinen erwarteten zukünftigen Nutzen zuordnet. Realisiert werden kann so eine Funktion beispielsweise durch ein neuronales Netz, wie Abbildung 4.1 es zeigt. Die Parameter werden am Ende des Spiels – nachdem die entsprechenden Bewertungen für den Endzustand (Sieg oder Niederlage) zurückgegeben wurden – aktualisiert. Das Ziel hierbei ist es die Evaluierungsfunktion zu trainieren, um die Wahrscheinlichkeit des Gewinnes genauer zu approximieren.
TD(λ) wurde bisher vornehmlich bei Brettspielen wie Schach und Backgammon angewendet. Diese haben den Vorteil, dass das Brett und die Spielregeln leicht zu simulieren und überwachen sind. Außerdem sind Input und Performance begrenzt und wohl definiert, und das Spielergebnis leicht zu bestimmen. Zudem haben beide Spieler in Bezug auf den gegenwärtigen Spielzustand das gleiche Wissen (es gibt keine

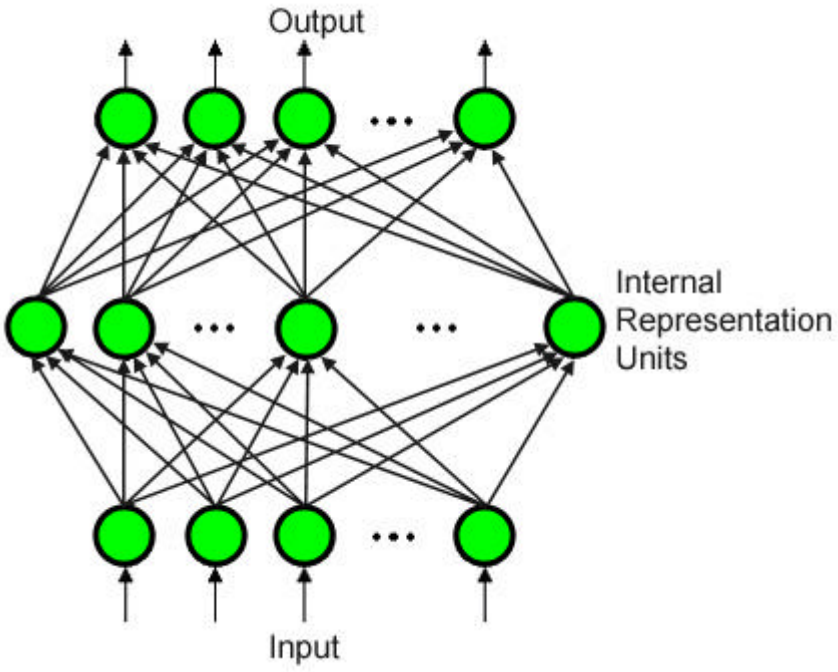

Abb. 4.1: Ein neuronales Netz besteht aus einer Eingabeschicht, internen Repräsentationseinheiten (Zwischenschicht), einer Ausgabeschicht. Die verschiedenen Schichten sind durch und gewichtete Kanten miteinander verbunden. Aus den Daten der Eingabeschicht werden in den Zwischenschichten Vorhersagen berechnet und über die Ausgabeschicht ausgegeben. Gelernt wird durch Änderung der Kantengewichte.

Bereiche, die nur einer der Spieler sehen kann). Dennoch erfordert das Spiel für den Expertenlevel eine komplexe und hoch entwickelte Spielweise des Agenten. Das Ergebnis des Lernprozesses und seine Qualität lassen sich also dementsprechend sehr gut nachvollziehen und bewerten.

Der wahrscheinlich bemerkenswerteste Erfolg von TD(λ) ist Tesauros TD-Gammon [2]. Es basiert auf einem neuronalen Netzwerk, lernte in simuliertem Spiel gegen sich selbst, wobei ohne jegliches Wissen über das Spiel gestartet wurde, und übertraf letztendlich alle bisherigen Backgammon-Computerspieler und bewies sich auch gegenüber den besten menschlichen Spielern. Es ging sogar so weit, dass Weltklassespieler die Eröffnungszüge des TD-Gammon übernahmen, da sie sich als besser als die bisher standardmäßig angewendeten bewiesen [2]. Diese Erfolge wurden dadurch ermöglicht, dass der TD-Algorithmus im Gegensatz zu den bisherigen Ansätzen bei Computerspielen – dem Design einer Bewertungsheuristik von menschlichen Experten und überwachtem Lernen anhand einer Spieldatenbank – keinerlei Vorwissen über das Spiel und keinen menschlicher Lehrer benötigt, wodurch der Agent nicht von der Auffassung der menschlichen Spieler beeinflusst wurde.

4.2 Reinforcement-Lernen

Temporal-Difference-Lernen ist eine Erweiterung des Reinforcement-Lernens. Für dieses Lernen durch Verstärkung erhält der Agent eine Eingabe, die das Wissen über den derzeitigen Zustand s(t), in dem sich seine Umwelt befindet, repräsentiert. In jedem Zustand gibt es eine Menge A von Aktionen a(t), die ausgeführt werden können. Um unter diesen möglichen Aktionen sinnvoll eine auswählen zu können, besitzt der Agent eine zeitabhängige Strategie π (engl. policy), welche die aktuellen Zustände und möglichen Aktionen auf einen numerischen Wert abbildet [5]:

$$\pi_t : S \times A \rightarrow [0, 1]$$

Die Aktion mit dem besten Wert wird ausgewählt und durchgeführt, da sie den höchsten Nutzen verspricht. Direkt nachdem die Aktion beendet ist, bekommt der Agent eine Belohnung r(t) (eng. reward). Diese teilt ihm mit, wie gut oder schlecht seine Handlung war, und er ändert seine Strategie dementsprechend. Es gibt aber auch viele Fälle, in denen die Belohnung erst verzögert nach einer Reihe von Aktionen erfolgt. Sie muss dann entsprechend dem Credit-Assignment-Problem [9] auf die einzelnen Aktionen verteilt werden. Auf beide Weisen werden Erfahrungen gesammelt, mit dem Ziel die besten Aktionen durchzuführen und so die Belohnungen zu maximieren. Abbildung 4.2 stellt den Lernprozess dar.
Diese Art des Lernens bietet im Gegensatz zum Überwachten Lernen, bei dem ein Lehrer nach jeder berechneten Ausgabe den eigentlich korrekten Wert (der hier bekannt ist) und den Fehler zur Korrektur mitteilt, den Vorteil, dass alleine – ohne intelligenten Lehrer – nur von der eigenen Erfahrung gelernt werden kann.

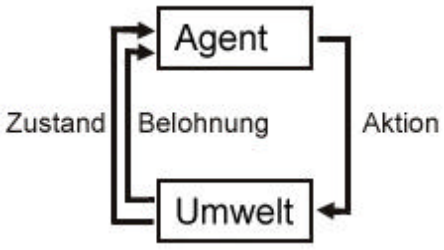

Abb. 4.2: Reinforcement Learning.

Abb. 4.3: TD-Learning.

Trotzdem gab es jahrelang nur wenig Erfolg beim Lösen großer, komplexer Probleme, da beim Reinforcement-Lernen mit verzögerter Belohnung die zeitliche Verteilung der Belohnung auf die einzelnen Aktionen schwierig blieb, und die traditionellen Anwendungen des Reinforcement-Lernens auf Tabellen zum Nachschlagen oder lineare Evaluationsfunktionen beschränkt war [2]. Die Entwicklung der Temporal-Difference (TD) Methoden von Sutton löste dieses Problem.

4.3 Der TD(λ)-Algorithmus

Die grundlegende Idee der TD-Methoden ist, dass das Lernen auf dem Unterschied zwischen zeitlich aufeinander folgenden Vorhersagen basiert [2]. Das Ziel des Lernens ist es folglich die Vorhersage für den gegenwärtigen Input der nächstfolgenden Vorhersage anzunähern.
Deshalb gibt es bei jeder TD-Methode – dargestellt in Abbildung 4.3 – ein heuristisches TD-Error-Signal, welches für jeden Zeitschritt anhand des Unterschiedes zwischen zwei aufeinander folgenden Vorhersagen definiert wird und das Lernen lenkt. Beim TD(λ)-Algorithmus zum Trainieren eines neuronalen Netzwerkes wird außerdem jeder TD-Error auch an die vorhergegangenen Schätzungen vorhergegangener Zustände weitergegeben, die so gleichzeitig auch korrigiert werden. Der λ-Parameter steuert dabei wie stark dieses Feedback weitergegeben wird. Er wird später ausführlicher erklärt.
Sei nun $x_1, x_2, \ldots x_{N-1}, x_N$ eine Sequenz von Zuständen eines Spieles. Zur Bewertung der einzelnen Zustände wird eine heuristische Evaluationsfunktion $J(x, w)$ verwendet:

$$J(x_i, w) = x_{i1} \cdot w_1 + x_{i2} \cdot w_{2 + \ldots +} x_{ik} \cdot w_k,$$

Hierbei ist $w = (w_{1,\ldots,} w_k)$ ein Gewichtsvektor, der mit dem Merkmalsvektor $(x_{i1,\ldots,} x_{ik})$ des zu bewertenden Zustandes x_i multipliziert wird.
Das Spielergebnis (Reward des letzten Spielzugs) $r(x_N)$, das der Lernende am Ende des Spieles erhält, ist entweder -1 für verloren, 1 für gewonnen oder 0 für unentschieden. Um den Wertebereich der übrigen Zustände entsprechend anzupassen, verwendet man eine Funktion $r(x, w)$ mit r: X × W → [-1, 1], beispielsweise eine hyperbolische Tangentialfunktion:

$$r(x, w) = \tanh(\beta \cdot J(x, w)),$$

wobei β eine kleine Konstante ist, welche die Tangentialfunktion streckt, damit sie nicht zu steil ist und eine ausreichende Spanne des Wertebereiches abdeckt. Im Folgenden wird größeren negativen Werten die -1 zugeordnet, größeren positiven Werten die 1, und die Spanne dazwischen als *„**aktiver Bereich**“* bezeichnet. Die Tangentialfunktion und der „aktive Bereich“ sind auf Abbildung 4.4 zu sehen.
Aus diesen Werten können nun die temporalen Differenzen d_i bestimmt werden. Jedes d_i ist definiert als der Unterschied zwischen den Belohnungen, die für die Aktionen i+1 und i vorhergesagt wurden. Die Differenz zwischen Spielergebnis und Vorhersage beim vorletzten Spielzug ist d_{N-1}. Für jeden Spielzug i = 1,…,N-2 gilt folglich

$$d_i = r(x_{i+1}, w) - r(x_i, w)$$

Und für N-1

$$d_{N-1} = r(x_N, w) - r(x_{N-1}, w)$$

Bei einer guten Evaluationsfunktion sollten die d_i, i = 1,…,N-1, möglichst klein sein, denn je kleiner die Änderungen zwischen den Belohnungen ist, desto genauer sind die getroffenen Belohnungs-Vorhersagen. Ausnahmen bilden hierbei die vom Gegner verursachten Fehler.

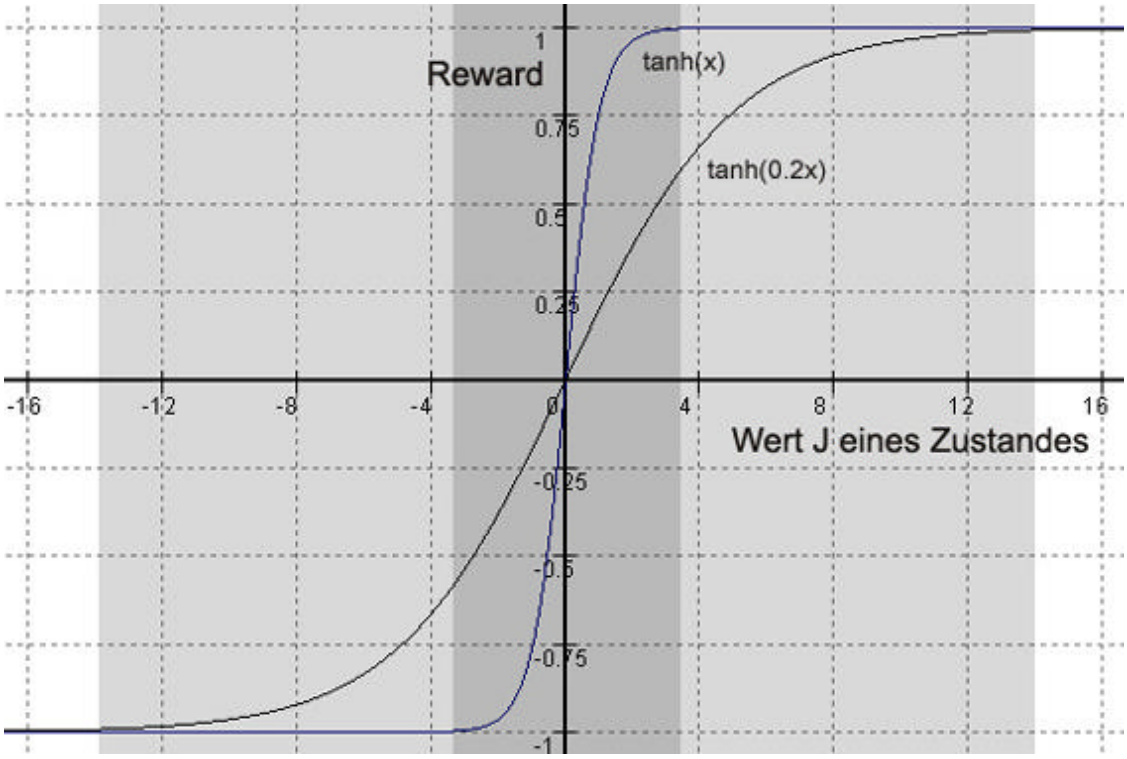

Abb. 4.4: Berechnung des Rewards r(x, w) = tanh(β J(x, w)), mit β = 0 und β = 0,2. Hellgrau: „aktiver Bereich" von tanh(0,2x); mittelgrau: „aktiver Bereich" von tanh(x).

Aus diesen Werten können nun die temporalen Differenzen $\boldsymbol{d_i}$ bestimmt werden. Jedes $\boldsymbol{d_i}$ ist definiert als der Unterschied zwischen den Belohnungen, die für die Aktionen i+1 und i vorhergesagt wurden. Die Differenz zwischen Spielergebnis und Vorhersage beim vorletzten Spielzug ist $\boldsymbol{d}_{N-1}$. Für jeden Spielzug i = 1,…,N-2 gilt folglich

$$\boldsymbol{d_i = r(x_{i+1}, w) - r(x_i, w)}$$

Und für N-1

$$\boldsymbol{d_{N-1} = r(x_N, w) - r(x_{N-1}, w)\ .}$$

Bei einer guten Evaluationsfunktion sollten die $\boldsymbol{d_i}$, i = 1,…,N-1, möglichst klein sein, denn je kleiner die Änderungen zwischen den Belohnungen ist, desto genauer sind die getroffenen Belohnungs-Vorhersagen. Ausnahmen bilden hierbei die vom Gegner verursachten Fehler.
Die Gewichte $\boldsymbol{w}$ sollen nun mit Hilfe der bisher bestimmten Werte und der folgenden Formel so justiert werden, dass sie die Größe der $\boldsymbol{d_i}$ verringern. Dadurch lernt der Agent, da so die Belohnung eines Zustandes an die des darauf folgenden angepasst wird. Nach Ende des Spiels und Berechnung der $\boldsymbol{d_i}$ nimmt der TD(λ)-Algorithmus ein Update der Gewichte vor, und zwar nach der folgenden Formel:

$$\boldsymbol{w := w + \alpha \sum_{i=1}^{N-1} \nabla r(x_i, w) \left[\sum_{j=i}^{N-1} \lambda^{j-i} d_i \right].} \qquad \textbf{(1)}$$

Der Parameter α kontrolliert die Lernrate, ist also dafür zuständig wie stark die Gewichte geändert werden, und sollte daher positiv sein und sich im Laufe des Lernens nach und nach Null annähern.

$$\nabla r(x_i, w) = \left(\frac{\partial r(x_i, w)}{\partial w_1} , \ldots, \frac{\partial r(x_i, w)}{\partial w_1} \right)$$

ist der Vektor der partiellen Ableitungen von $r(x_i, w)$ bezüglich w. Angenähert werden kann sie beispiels-weise (mit hinreichend kleinem δ) durch

$$\frac{\partial r(x_i, w)}{\partial w_j} \approx \frac{r(x_i, w_j + \delta) - r(x_i, w_j)}{\delta} .$$

Folglich gilt für die einzelnen Parameter w_j

$$w_j := w_j + \alpha \sum_{i=1}^{N-1} \frac{\partial r(x_i, w)}{\partial w_j} \left[\sum_{m=i}^{N-1} \lambda^{m-i} d_m \right].$$

Der Parameter λ kontrolliert wie weit die temporalen Differenzen das Veränderung der Gewichte beeinflussen. Mit λ = 0 sieht Gleichung (1) folgendermaßen aus:

$$w := w + \alpha \sum_{i=1}^{N-1} \nabla r(x_i, w) d_i$$

$$= w + \alpha \sum_{i=1}^{N-1} \nabla r(x_i, w) \left[r(x_{i+1}, w) - r(x_i, w) \right] ,$$

und für λ = 1:

$$w := w + \alpha \sum_{i=1}^{N-1} \nabla r(x_i, w) \left[r(x_N) - r(x_i, w) \right] .$$

Mit λ = 0 werden die Gewichte so justiert, dass die Differenz zwischen $r(x_i, w)$ und $r(x_{i+1}, w)$ verkleinert wird, so dass sich die vorhergesagte Belohnung für Aktion i weniger von der von i+1 unterscheidet. Im Unterschied dazu wird mit λ = 1 die für Aktion i vorhergesagte Belohnung in Richtung der Belohnung des letzten Spielzuges geändert. Die λ-Werte zwischen Null und Eins interpolieren zwischen diesen beiden Verhaltensweisen. Abbildung 4.5 veranschaulicht dies. Man kann auch sagen, dass für kleine λ-Werte die direkt folgenden Aktions-Belohnungen als Grundlage für das Update dienen, und für große λ-Werte die des Spielendes. Deshalb sollte man, wenn die Evaluationsfunktion schon recht gut funktioniert, einen kleinen Werte für λ wählen (0,3 bis 0,7), und falls nicht einen Wert nahe Eins (etwa 0,95 bis 1) [3].
Abbildung 4.6 zeigt den vollständigen TD-Algorithmus als Pseudocode.
In einigen Fällen kann es notwendig werden die Gewichte w zu normieren, da sonst durch die ständige Addition der Änderungen während der Updates die Werte der heuristischen Evaluationsfunktion den „aktiven Bereich“ der Wertefunktion verlassen können, d.h. es gibt dann nur noch Ergebnisse von -1 oder 1.

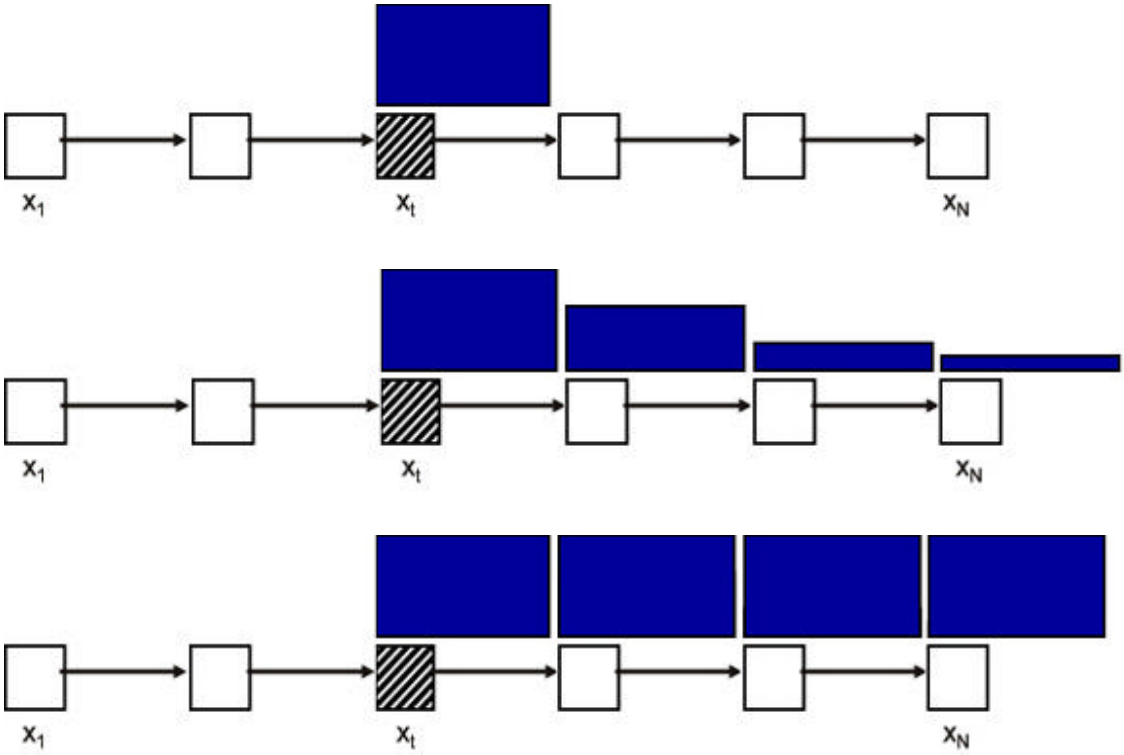

Abb. 4.5: TD- Learning: der λ- Parameter. Oben: λ = 0; Mitte: 0<λ<1, unten: λ =1. Bei λ = 0 wird beim Update der Gewichte nur die direkte Differenz zwischen den Belohnungen zweier aufeinander folgender Zustände betrachtet. Bei 0<λ<1 werden die Differenzen zu allen nachfolgenden Zuständen benutzt, die näher liegenden stärker als die späteren. Und bei λ =1 ist die Differenz zum letzten Zustand ausschlaggebend.

```
public void learnTD() {
  initialize w
  while (improve game play) {
    initialize x
     while  (x is not terminal) {
      a = the action for x, that has the highest reward
      take action a, observe next state x'
      compute (and store till game end) temporal difference td:
          td = r(x',w) – r(x,w)
      and partial derivatives pd:
          pd = △r(x',w)
      x = x'
    }
    r = terminal reward (-1 for loss, 0 for draw ,1 for win)
    compute temporal difference
    update w
  }
}
```

Abb. 4.6: TD(λ)- Algorithmus mit Zustand x, Aktion a, Belohnung r, Gewicht w.

4.4 Die lernenden Skip-Bo-Spieler: Anwendung des TD-Lernens

Dieser Abschnitt geht auf die Implementierung der lernenden Agenten A^{TD} und A^{Lea} und die Lern-Experimente ein.
Die grundlegende Routine der Spielzüge unterscheidet sich hier wesentlich von denen der in Abschnitt 3 vorgestellten regelbasierten Skip-Bo-Spieler. Bei diesen ist die Reihenfolge, in welcher die Karten der verschiedenen Spielstapel auf Spielbarkeit getestet und ausgespielt werden, vorgegeben. Bei den lernenden Agenten hingegen werden nun alle Möglichkeiten zugleich getestet, und dann anhand der Evaluationsfunktion die beste mögliche Aktion ausgewählt und ausgeführt, wie in Abbildung 4.7 beschrieben.

```
public void makeYourMove() {
  while (not yet done discard action) {
    a = the action, that has the highest reward
    x' = the state a is leading to
    compute (and store in vector) temporal difference*
    compute (and store in vectors) partial derivatives of x'
    x = x'
  }
}
```

Abb. 4.7: Grundlegende Routine der lernenden Agenten.

(*) except for very first state of the game

Sobald das Spielende erreicht ist, wird die entsprechende Endbelohnung zurückgegeben (1 für Sieg, -1 sonst, da es bei Skip-Bo kein unentschieden gibt) und die temporale Differenz zum vorherigen Zustand gespeichert. Anschließend werden – wie im vorhergehenden Abschnitt beschrieben – die Gewichte w angepasst.
Damit ein Agent das Skip-Bo-Spielen lernen kann, ist es notwendig die Parameter zur Bewertung der Zustände zu bestimmen, die dann durch die w_i verschieden stark gewichtet werden. Das heißt, es müssen einzelne Zustands-Merkmale extrahiert werden, die zusammen- genommen jeden möglichen Zustand sinnvoll und möglichst vollständig beschreiben. Von diesen Merkmals-Parametern hängt es ab, ob die Bewertung ausreichend genau ist, um damit eine erfolgreiche Spielweise erlernen zu können.
Das Ergebnis meiner Experimente, bei denen ich verschiedenste Parameter in unterschiedlichen Kombinationen getestet habe, sind zwei selbstlernende Agenten. Diese beiden – A^{TD} und A^{Lea} – unterscheiden sich in der Anzahl und Art der verwendeten Parameter, und dementsprechend auch in ihrer Spielweise und deren Erfolg.

4.4.1 TD lernender Agent A^{TD}

A^{TD} ist ein selbstlernender Skip-Bo-Agent, der über die in Abbildung 4.8 aufgeführten 14 Parameter zur Bewertung eines jeden Zustandes verfügt.
Diese Parameter repräsentieren Zustands-Merkmale, aus denen sich die Kniffe aus Abschnitt 2.3 darstellen lassen:

Kniff 1 (*Vorratsstapel favorisieren*) wird durch 1. repräsentiert, sobald A^{TD} ein ausreichend großes (negatives – da es ja möglichst wenig Karten sein sollen) Gewicht gelernt hat.

1.	Die Anzahl der Karten, die auf dem Vorratsstapel liegen.
2.	Minimaler „Abstand“: Anzahl der Karten, die mindestens noch gelegt werden müssen, bevor die oberste Karte des Vorratsstapels gespielt werden kann
3.	Durchschnittliche „Abstand“ aller vier Aufbaustapel zum Vorratsstapel
4.	Anzahl der Karten auf der Hand, die keine Skip-Bos sind
5.	Anzahl der Skip-Bos auf der Hand
6.	Anzahl aller Karten auf den Ablagestapeln
7.-11.	Nutzung der Ablagestapel entsprechend Kniff 3
12.	Anzahl der vom Agenten auf die Aufbaustapel gelegten Karten
13.	Anzahl der Karten, die auf dem gegnerischen Vorratsstapel liegen
14.	Anzahl der Karten, die mindestens noch gelegt werden müssen, bevor der Gegner die oberste Karte seines Vorratsstapels spielen kann

Abb. 4.8: A^{TD}s Parameter.

Für Kniff 2.1 (*Rangfolge der Aufbaustapel*) stehen die Parameter 2 und 3, denn durch das Legen einer Karte auf den besten Aufbaustapel wird die minimale und die durchschnittliche Differenz verringert, für jeden anderen nur die durchschnittliche Differenz.

Kniff 2.2 (*Rangfolge von Hand und Ablagestapel*) ergibt sich aus der Kombination von 4, 5 und 6, wobei die Trennung zwischen 4 und 5 für Kniff 4 (*Skip-Bo*) notwendig ist.

Für Kniff 3 (*Gebrauch der Ablagestapel*) werden fünf Parameter gebraucht:
Zwei davon betreffen die untersten Karten der Ablagestapel, also die Karten, die auf einen leeren Stapel gelegt wurden. Der erste ergibt sich aus dem durchschnittlichen Wert dieser Karten. Damit lässt sich lernen, ob es besser ist, einen Ablagestapel mit einer hohen oder einer niedrigen Karte zu beginnen. Der zweite Parameter misst, wie viele dieser Karten einen gleichen Wert haben, da es wenig Sinn macht jeden Ablagestapel mit dem gleichen Wert zu anzufangen (z.B. ist bei 7, 7, 7, 8 das Ergebnis 3).
Der dritte Parameter gibt den prozentualen Anteil der Karten an, die auf eine Karte mit demselben Wert gelegt wurden, und der vierte den der Karten, die als „Straße“ liegen. Der fünfte Parameter gibt schließlich den Anteil der Karten zurück, die nicht unter die vorherigen Ablagepunkte fallen.

Kniff 5 (*Beobachten des Gegners*) ist dargestellt durch die Parameter 13 und 14.

Der 12. Parameter betrifft keinen der Kniffe direkt, wurde aber notwendig, da der Agent sonst (auch mit Lernen) dazu neigte einen Spielzug zu früh zu beenden und dadurch keine guten Spielergebnisse lieferte.

Mit diesen 14 Parametern, $\lambda = 0{,}7$, $\alpha = 10^{-7}$ und einer Anfangsgewichtung von Null, d.h. keinerlei Vorwissen über das Spiel, startet der Lernprozess. Da der Agent zu diesem Zeitpunkt noch nicht weiß, wie er spielen soll, verliert er gegen die anderen Spieler – A^B und A^M –zu oft, um dabei effektiv lernen zu können. Deshalb wird ein Lern-Spiel simuliert, in dem A^{TD} gegen sich selbst antritt, bzw. gegen eine zweite A^{TD}-Instanz mit gleichem Wissen. Abbildung 4.9 zeigt einen auf diese Weise erzielten Lernprozess: es wurde abwechselnd ein Lernschritt gemacht, und dann die gelernte Gewichtung in

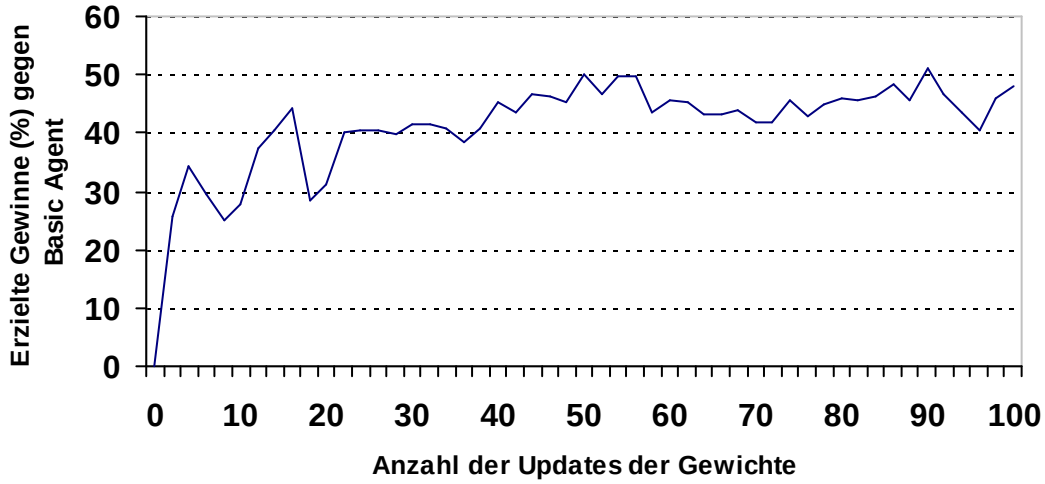

Abb. 4.9: A^{TD}s Lernprozess. Getestet wurde jeweils in 1000 Spielen gegen A^{B}.

jeweils 1000 Spielen gegen A^{B} getestet. Hierbei gewann A^{TD} schon nach dem 50ten Update 501 der 1000 Spiele.
Während der Updates im simulierten Spiel gegen sich selbst lässt sich beobachten, dass die Anzahl der benötigten Kartenzüge pro Spiel abnimmt je besser die Parameter sind, d.h. die Spiele werden immer schneller entschieden während die Bewertung der Zustände bezüglich des erwarteten Gewinns immer genauer wird.
Nach einigen Updates im Spiel gegen sich selbst, teilweise mit kleineren λ- und α-Werten, entstand eine Gewichtung der Parameter, welche sich auch durch weiters Lernen im Spiel gegen andere Agenten nicht merklich verbessern ließ. Diese wurde für die weiteren Untersuchungen als abschließende Gewichtung verwendet.
Doch selbst mit den besten Gewichtungen konnte A^{TD} gegen A^{B} keine höhere Gewinnquote als knapp 50% (bei 10.000 Spielen) erzielen. Grund hierfür ist, dass der Zustand des Spielfeldes bei diesem Kartenspiel nicht alle benötigen Informationen enthält. Denn man kann über den Zustand zwar Aussagen über die verschiedenen Kartenstapel und -werte machen, aber nicht über den spezifischen Kartenzug. Das bedeutet, man kann messen, wie viele Karten sich auf welchem Stapel befinden, welchen Wert sie haben, wie viele Karten noch gelegt werden müssen, damit die oberste Karte des Vorratsstapels gelegt werden kann, und wie viele Skip-Bos sich auf der Hand befinden. Was fehlt, ist beispielsweise eine klare Aussage darüber, ob es in der Situation Sinn macht einen Skip-Bo zu spielen, und darüber, ob es besser ist eine Karte von der Hand oder von einem Ablagestapel zu spielen. Deshalb wurde der zweite lernende Agent entwickelt.

4.4.2 Learning Agent A^{Lea} - Bewertung der Aktion

Bisher wurden immer ausschließlich die Spielzustände bewertet, nicht die einzelnen Aktionen selbst. Bei diesem Kartenspiel ist es jedoch einfacher eine Aktion auszusuchen, wenn man diese direkt bewertet, und nicht nur den Zustand, der dadurch entsteht. Diese Methode wird in Abbildung 4.10 dargestellt. Auf diese Weise kann genau unterschieden werden von wo eine Karte genommen wird (ist es besser zuerst von der Hand oder zuerst vom Vorratsstapel zu spielen) oder wohin eine Karte am besten gelegt wird (auf den vorteilhaftesten Aufbaustapel, einen anderen Aufbaustapel oder einen Vorratsstapel).

A^{Lea} basiert auf dieser Methode, die hier zwar ein besseres Ergebnis liefert, aber auch Probleme mit sich bringt, die im Folgenden beschrieben und gelöst werden.
A^{Lea} benutzt 33 Parameter, um den Wert eines Kartenzuges zu bestimmen. Die Parameter sind in vier Gruppen aufgeteilt:

1. den Ort, von dem die Karte genommen wird (***Quelle***),
2. den, wohin sie gelegt wird (***Ziel***),
3. spezielle Parameter für den Gebrauch der Skip-Bo-Karten
4. und für die Frage, ob der Gegner durch den Kartenzug in der Lage sein wird die oberste Karte seines Vorratsstapels zu spielen.

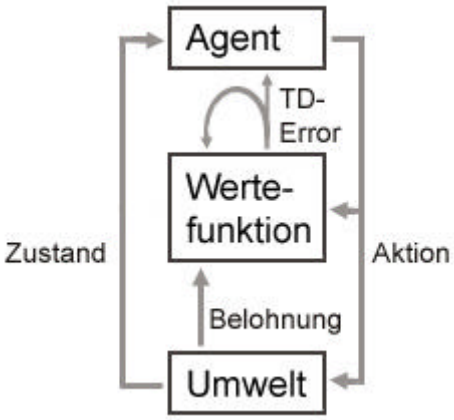

Abb. 4.10: TD- Learning mit Bewertung der Aktion (vgl. Abb. 4.2 in Abschnitt 4.2)

Die Parameter-Gruppe Ziel ist weiter unterteilt in die Bereiche ***Ziel: Aufbaustapel*** und ***Ziel: Ablagestapel***. Letzteres ist nochmals aufgeteilt in ***Strategie***, ***Wert der Karte*** und ***Anzahl***. Abbildung 4.11 zeigt die genauere Aufgliederung dieser vier Parameter-Gruppen. Für die Parameter wird von nun an folgende Konvention verwendet (Reihenfolge innerhalb der Gruppen wie in Tabelle der Abbildung 4.11):

1. Die zu Quelle gehörenden Parameter werden abgekürzt durch:
   $Q^{Hand(Ziel:Aufbau)}$, $Q^{Hand(Ziel:Ablage)}$, Q^{Vorrat}, Q^{Ablage} und Q^{ABonus}.

2. Die Ziel-Parameter sind:
   - $Z^{Aufbau:bester}$, $Z^{Aufbau:nBester}$, $Z^{Aufbau:nBester2}$,
   - $Z^{Ablage:Strat:leer}$, $Z^{Ablage:Strat:gleich}$, $Z^{Ablage:Strat:Straße}$, $Z^{Ablage:Strat:aufN}$, $Z^{Ablage:Strat:aufH}$,
   - $Z^{Ablage:Wert:1}$, $Z^{Ablage:Wert:2}$,…, $Z^{Ablage:Wert:12}$,
   - $Z^{Ablage:Anzahl}$

3. Die speziellen Parameter für Skip-Bos:
   $SB^{gleicherWert}$, $SB^{Folgekarte}$, SB^{nur}

4. Die Parameter bezüglich des Gegners:
   G^{kann}, $G^{LeaKannVerhindern1}$, $G^{LeaKannVerhindern2}$, $G^{LeaKannVerhindern3}$.

Alle Parameter (bis auf $Z^{Ablage:Anzahl}$) liefern einen Rückgabewert von `1` oder `0`, bzw. `-1` oder `0`, wobei die `0` immer bedeutet, dass der betreffende Fall nicht erfüllt ist. So erhält die Aktion „lege eine Eins auf einen (leeren) Aufbaustapel“ als erster Zug eines Spiels eine `1` für die Parameter $Q^{Hand(Ziel:Aufbau)}$ und $Z^{Aufbau:bester}$. Und je nachdem, ob der Gegner dadurch eine Karte von seinem Vorratsstapel spielen kann, hat G^{kann} den Wert `-1` und eventuell auch einer der $G^{LeaKannVerhindern}$-Parameter `1`. Aber alle sonstigen Werte sind `0`. Ein weiteres Beispiel für die Funktionsweise der Parameter sei die Aktion: „ Lege eine von zwei Zwölf aus der Hand auf einen leeren Ablagestapel“. Für diese Handlung liefern die Parameter $Q^{Hand(Ziel:Ablage)}$, $Z^{Ablage:Strat:leer}$ und $Z^{Ablage:Wert:12}$ den Wert `1`, $Z^{Ablage:Anzahl}$ eine `2`, und alle anderen `0`.
Auf diese Weise kann jede Art von Spielzug differenziert betrachtet und bewertet werden, und das Update der Gewichte am Ende des Spiels korrigiert entsprechend den Wert der einzelnen Kartenzüge.

<table>
<tr><th>Quelle</th><th>Skip-Bo-Karten</th><th>Gegner</th></tr>
<tr><td>Hand, wenn Ziel Aufbaustapel: $Q^{Hand(Ziel:Aufbau)}$</td><td>Karte mit entsprechendem Wert spielbar?*: $SB^{gleicherWert}$</td><td rowspan="5">Karte seines Vorratsstapels spielbar?*: G^{kann}
Kann A^{Lea} diese Karte spielen? (Drei Parameter für verschiedene Fälle): $G^{LeaKannVerhindern1}$, $G^{LeaKannVerhindern2}$, $G^{LeaKannVerhindern3}$</td></tr>
<tr><td>Hand, wenn Ziel Ablagestapel: $Q^{Hand(Ziel:Ablage)}$</td><td>Folgekarte spielbar?*: $SB^{Folgekarte}$</td></tr>
<tr><td>Vorratsstapel: Q^{Vorrat}</td><td rowspan="3">Nur noch Skip-Bos auf der Hand?: SB^{nur}</td></tr>
<tr><td>Ablagestapel: Q^{Ablage}</td></tr>
<tr><td>+Ablagestapelbonus, wenn darunter liegende Karte dann auch spielbar: Q^{ABonus}</td></tr>
</table>

<table>
<tr><th colspan="4">Ziel</th></tr>
<tr><th rowspan="2">Aufbaustapel</th><th colspan="3">Ablagestapel</th></tr>
<tr><th>Strategie</th><th>Wert der Karte</th><th>Anzahl</th></tr>
<tr><td>Bester Stapel: $Z^{Aufbau:bester}$</td><td>Leerer Stapel: $Z^{Ablage:Strat:leer}$</td><td rowspan="5">12 Parameter, die jeweils nur für den entsprechenden Wert eine 1 liefern: $Z^{Ablage:Wert:1}$, $Z^{Ablage:Wert:2}$, …, $Z^{Ablage:Wert:12}$</td><td rowspan="5">Wie oft ist der jeweilige Wert auf der Hand vorhanden? ** $Z^{Ablage:Anzahl}$</td></tr>
<tr><td rowspan="2">Nicht bester, da nicht möglich: $Z^{Aufbau:nBester}$</td><td>Auf Karte gleichen Wertes: $Z^{Ablage:Strat:gleich}$</td></tr>
<tr><td>Als „Straße“: $Z^{Ablage:Strat:Straße}$</td></tr>
<tr><td rowspan="2">Nicht bester, obwohl möglich*: $Z^{Aufbau:nBester2}$</td><td>Auf niedrigere Karte: $Z^{Ablage:Strat:aufN}$</td></tr>
<tr><td>Auf höhere Karte*: $Z^{Ablage:Strat:aufH}$</td></tr>
</table>

Abb. 4.11: Die Parameter des lernenden Agenten A^{Lea} aufgeteilt nach der Quelle der Karte (Ort, von dem sie genommen wird), dem Ziel der Karte (wohin sie gelegt wird), und den Parametern, welche die Skip-Bo-Karten und die generische Situation betreffen. Die Ziel-Parameter sind weiter unterteilt nach Aufbau- und Ablagestapel, wobei Ablagestapel aus drei Untergruppen besteht.

Rückgabewerte: [0,1]; [-1,0] für (*); [0,…,5] für (**). Hierbei bedeutet 0 immer,

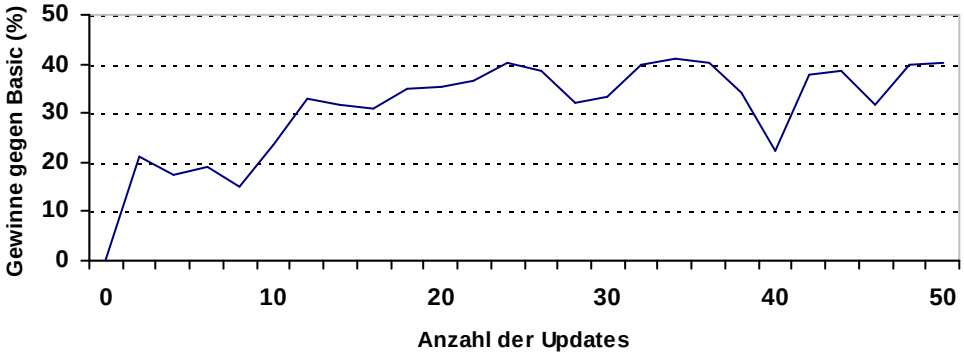

Abb. 4.12: A^{Lea}s Lernprozess mit der Anfangsgewichtung von Null.

Als Problem dabei zeigte sich, dass beim Lernen mit den gleichen Startwerten wie bei A^{TD} ($\lambda = 0{,}7$, $\alpha = 10^{-7}$ und Anfangsgewichte von Null) auch nach 300 Updates die Gewinnquote von 40% gegen A^{B} nicht übertraf. Dabei gewann A^{Lea} gegen A^{B} schon nach 24 Updates 402 von 1000 Spielen, wie Abbildung 4.12 zeigt. Der Grund dafür ist, dass für einige Parameter negative Vorzeichen gelernt wurden, obwohl logischerweise alle positiv sein müssen. So waren bei dem oben genannten Spiel (nach den 24 Updates) unter anderem die folgenden Gewichte negativ: $Q^{Hand(Ziel:Aufbau)}$, $Q^{Hand(Ziel:Ablage)}$, $SB^{gleicherWert}$, $Z^{Aufbau:nBester}$, $Z^{Aufbau:nBester2}$. Das heißt, es wurde als negativ ausgelegt eine Karte von der Hand zu spielen. Außerdem wurde durch die negative Gewichtung von $SB^{gleicherWert}$ gefördert, dass Skip-Bos gespielt wurden, obwohl eine Karte mit dem entsprechenden Wert vorhanden und spielbar war. Und weil die Gewichtung von $Z^{Aufbau:nBester}$ und $Z^{Aufbau:nBester2}$ negativ war, wurde Kniff 2.1 (*Rangfolge der Aufbaustapel*) nicht genutzt. Durch diese und weitere negative Gewichtungen wurde A^{Lea}s Spielweise nicht so gut, wie sie eigentlich hätte werden könnten.
Die Lösung hierfür ist es das Lernen mit einer Anfangsgewichtung ungleich Null zu starten, wobei man allerdings nicht vergessen sollte die Gewichte zu normieren. Denn ansonsten verlassen nach einigen Updates die Werte der heuristischen Evaluationsfunktion *J(x, w)* den „aktiven Bereich“ der Bewertungsfunktion *r(x, w)* – wie in Abschnitt 4.3 beschrieben. Dadurch gibt die Bewertungsfunktion dann für fast alle Zustände Ergebnisse von -1 oder 1 zurück, und der Agent spielt nur noch sehr schlecht.
Damit der Agent ordnungsgemäß und ohne Vorwissen lernen kann, sollte die gewählte Anfangsgewichtung jeder Aktion den gleichen Wert zuordnen. Denn der Agent kann nur dann unvoreingenommen sein, wenn jede Aktion neutral gegenüber den anderen Aktionen ist.
In diesem Fall setzt sich die Bewertung einer jeden Aktion aus den zuvor vorgestellten vier Parameter-Gruppen zusammen. Damit jede Aktion denselben Startwert erhält, müssen folglich die Parameter einer Gruppe das gleiche Gewicht haben. Dieses *Einheitsgewicht* für jede Gruppe sei *ew(<Gruppenbezeichner>)*, also beispielsweise für die Parameter-Gruppe Quelle: *ew*(Q). Da in der Gruppe Ziel Z^{Ablage} weitere Untergruppen enthält, muss hier das Einheitsgewicht zu gleichen Teilen auf die Untergruppen verteilt werden. Das bedeutet: $ew(Z) = ew(Z^{Aufbau}) = ew(Z^{Ablage}) = ew(Z^{Ablage:Strat}) + ew(Z^{Ablage:Wert}) + ew(Z^{Ablage:Anzahl})$. Des Weiteren muss das Einheitsgewicht der Gegner-Parameter gelten: $ew(G) = ew(Q) + ew(Z)$, da in den entsprechenden Fällen die Einheitswerte von Quelle und Ziel ausgeglichen werden sollen. Denn ein Zug ist nicht vorteilhaft, wenn der Gegner dadurch eine Karte seines Vorratsstapels spielen kann. Die Aktion hat dann also einen Gesamtwert von Null.
Jedoch, mit diesen Startgewichten lernt A^{Lea} – im Gegensatz zu Startgewichten von Null – meistens höhere Gewichtungen für Q^{Hand} als für Q^{Vorrat}, wodurch keine guten Spielergebnisse erreicht werden. Es werden einfach viel häufiger Karten von der Hand gespielt als vom Vorratsstapel, so dass dies als wichtiger bewertet wird und ein höheres Gewicht erhält.
Aber wenn man das Anfangsgewicht von Q^{Vorrat} erhöht und den Agenten somit nicht ohne Grundwissen über das Spiel sondern mit der in der original Spielanleitung mehrmals betonten Kenntnis, dass man auf jeden Fall immer zuerst die Karten des Vorratsstapels ausspielen soll, lernen lässt, gewinnt A^{Lea} schon nach sechs Updates im Spiel gegen sich selbst und darauf folgenden elf Updates im Spiel gegen A^{B} (mit $\lambda = 0{,}4$ und $\alpha = 10^{-9}$) ca. 60 % der Spiele gegen A^{B}, wie Abbildung 4.13 zeigt.

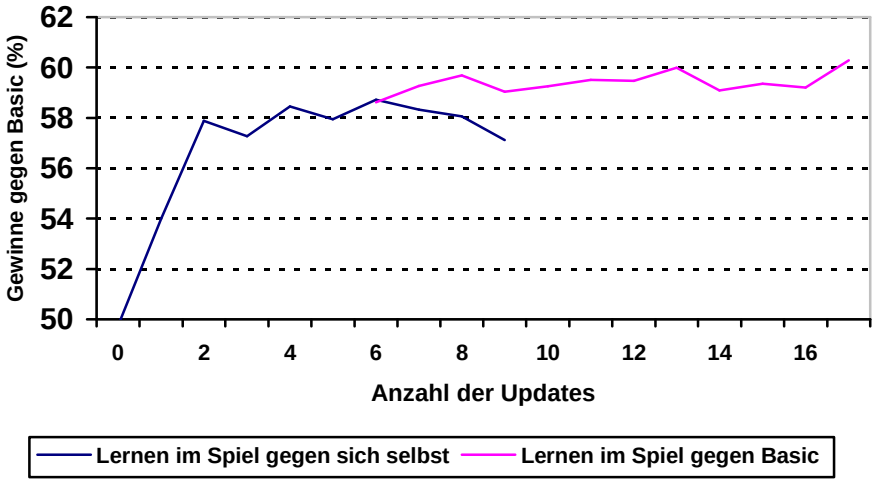

Abb. 4.13: A^{Lea}s Lernprozess gemessen in jeweils 10.000 Spielen gegen A^{B}.

5 Evaluation

Im Abschnitt über die Lernprozesse wurden bereits die lernenden Agenten A^{TD} und A^{Lea} mit A^{B} verglichen. In diesem Abschnitt werden die Fähigkeiten aller Spieler genauer untersucht.
Abbildung 5.1 zeigt, wie die Agenten im Spiel untereinander abschneiden, wobei als Grundlage jeweils 20.000 Spiele dienen. Nicht enthalten in dieser Tabelle ist der in Abschnitt 3.1 erwähnte Agent A^{B-}, welcher wie A^{B} ohne Ablagestrategie spielt. Dieser Agent erreicht im Spiel gegen A^{B} nur eine Gewinnquote von 42,11% und gegen A^{M} 27,27%. Diese Ergebnisse zeigen wie sehr Änderungen im Spielverhalten die Gewinnchancen verändern, und so auch kleine Defizite im Lernverhalten schon merkliche Auswirkungen haben.
A^{TD} und A^{Lea} wurden jeweils mit ihren abschließenden, gelernten Gewichten getestet.

	A^{M}	A^{TD}	A^{Lea}
A^{B}	35,66 : 64,34	51,9 : 48,1	41,02 : 58,98
A^{M}		65,7 : 34,3	56,38 : 43,62
A^{TD}			38,84 : 61,16

Abb. 5.1: Die Gewinnquoten der Spiele- Agenten untereinender. Zur Auswertung wurden die Ergebnisse von jeweils 20.000 Spielen genommen. Bei Wiederholung des Versuches können dennoch Abweichungen von bis zu zwei Prozentpunkten auftreten. A^{B} gewann gegen A^{M} 35,66% der Spiele (also A^{M} gegen A^{B} 64,34%), A^{TD} gegen A^{Lea} 38,84%, A^{B} gegen A^{TD} 51,9%, usw.

In den folgenden Abschnitten werden die Agenten und ihr Spiel im Einzelnen untersucht, beschrieben und bewertet. Dazu werden zum einen die Beobachtungen, die ich gemacht habe, während ich gegen sie spielte, benutzt. Des Weiteren wird eine Versuchsaufstellung mit drei gegeneinander spielenden Agenten verwendet, welche genauer zeigt, wie erfolgreich ein Spieler ist, bzw. wie er das Spiel seiner Gegner beeinflusst. Hierbei werden jeweils 30.000 Spiele betrachtet, und als Gegner jeweils A^{B} gewählt.

5.1 A^Bs Spielverhalten

Das Spielverhalten dieses Agenten ergibt sich aus der in Abschnitt 2.3 und 3.1 beschriebenen Programmierung. Dementsprechend spielt er immer alle Karten aus, die er spielen kann, und zwar in der Reihenfolge, in der er sie auf den Ablagestapeln und auf der Hand hat. Aber er beachtet, dass die Karten des Vorratsstapels immer zuerst gespielt werden müssen, verwendet eine klare Ablagestrategie und bevorzugt beim Ausspielen der Karten die Ablagestapel gegenüber der Hand, damit die unteren Karten der Stapel auch verwendet werden können, so dass er häufig längere Spielzüge machen kann und dadurch auch eine größere Chance hat den Vorratsstapel zu verkleinern. Dennoch werden die Karten unabhängig von dem Wert des Zuges (im Sinne von: was nützt er dem Ziel zu gewinnen) nicht beachtet. Ebenso werden auch Karten gespielt, die dem Gegner einen Vorteil verschaffen, da die gegnerischen Stapel überhaupt nicht betrachtet werden. Folglich kann man gegen ihn relativ leicht gewinnen, was auch die Statistik von Abbildung 5.1 zeigt.

5.2 A^Ms Spielverhalten

A^M spielt entsprechend der in den Abschnitten 2.3 und 3.2 beschriebenen Regeln. Somit nutzt er alle Kniffe und spielt meiner Meinung nach sehr gut, wie auch die Tabelle aus Abbildung 5.1 beweist.
Allerdings betreffen die Betrachtungen der gegnerischen Stapel immer nur die des direkt nachfolgenden Spielers (wie es aber bei A^{TD} und A^{Lea} auch der Fall ist).
Wie die Gewinnverhältnisse aussehen, wenn A^M gegen zwei gleichartige Gegner antritt, ist in Abbildung 5.2 (1) zu sehen. Die Ergebnisse zeigen, dass A^{B1}, der nach A^M an der Reihe ist mit einer Gewinnquote von 24,45% eindeutig schlechter abschneidet als A^{B2}, der erst danach spielt und 35% erreicht.
Abbildung 5.2 (2) zeigt wie die Gewinnverteilung ausfällt, wenn zwei A^Ms und ein A^B gegeneinander spielen. Man erkennt wieder, dass der Nachfolger eines A^Ms schlechtere Ergebnisse erzielt, denn A^{M1} ist mit 43,19% um einiges besser als A^M2 mit 30,33%.

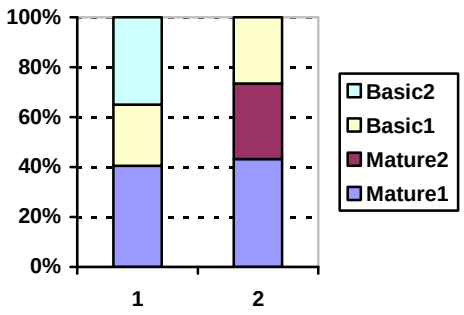

Abb. 5.2: Gewinnstatistik von A^M im Spiel dreier Agenten. Von unten nach oben: Bei (1) ist Spieler 1 A^M, Spieler 2 A^{B1} und Spieler 3 A^{B2}. Bei (2) folgen Spieler 1 A^{M1}, Spieler 2 A^{M2} und Spieler 3 A^B.

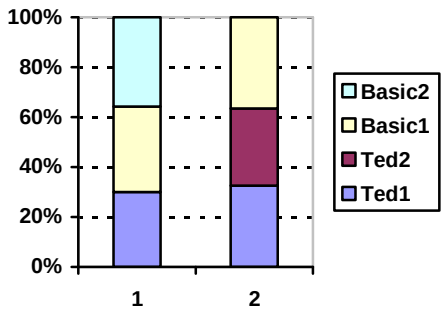

Abb. 5.3: Gewinnstatistik von A^{TD} im Spiel dreier Agenten. Bei (1) ist Spieler 1 A^{B1}, Spieler 2 A^{B2} und Spieler 3 A^{TD}. Bei (2) folgten Spieler 1 A^B, Spieler 2 A^{TD1} und Spieler 3 A^{TD2}.

5.3 A^{TD}s Spielverhalten

Bei A^{TD}s Spielverhalten fällt vor allem auf, dass er die Skip-Bos nicht immer sinnvoll einsetzt, sondern sie beispielsweise auf einen Aufbaustapel legt, auf dem sie ihm gar nichts nutzten. Außerdem ist seine Ablage-Strategie nicht immer konform zu Kniff 3, wodurch er seltener längere Spielzüge (also viele Karten hintereinander) spielen kann und das Spiel häufiger ins Stocken gerät. Er spielt auch oft Karten, die zum Vorteil des Gegners sind, d.h. so, dass der Gegner beim nächsten Zug die oberste Karte seines Vorratsstapels ausspielen kann.
Das Spiel von A^{TD} gegen zwei A^{B}s – zu sehen auf Abbildung 5.3 (1) – verdeutlicht noch einmal, dass A^{TD} nicht so gut spielt wie A^{B} (wie Abb.5.1 schon zeigt), aber beweist auch, dass er dennoch seinem direkt nachfolgenden Gegner einen kleinen Nachteil vermittelt: A^{B1} erreicht „nur“ eine Gewinnquote von 34,25%, A^{B2} dahingegen 35,75%; A^{TD} 30%.
Auch wenn zwei A^{TD}s gegen einen A^{B} antreten schneiden sie schlechter ab, aber A^{TD1} erreicht eine höhere Quote als A^{TD2}: A^{TD1}: 32,53, A^{TD2}: 30,91, A^{B}: 36,56.

5.4 A^{Lea}s Spielverhalten

A^{Lea}s Spielverhalten ist um einiges besser, als das von A^{TD}. Er nutzt die Ablagestapel besser, was sich darin äußert, dass er häufig sehr viele Karten pro Spielzug ausspielen kann. D.h. auch, dass seltener Situationen entstehen, in denen kein Spieler eine Karte auf die Aufbaustapel legen kann. Auch die Skip-Bos werden sehr viel sinnvoller genutzt, werden fast immer auf den besten Aufbaustapel gelegt, und behalten, wenn das Ausspielen A^{Lea} nicht weiterbringen würde. Zumeist behält er Karten, die im nächsten Zug des Gegners zum Ausspielen einer seiner Vorratsstapelkarten führen würden, bei sich, bis er selbst den entsprechenden Wert des gegnerischen Vorratsstapels besitzt.
Negativ fällt auf, dass A^{Lea} ab und zu Karten, die er legen könnte – ohne das dies negative Folgen hätte –, behält und dadurch manchmal sogar die Chance eine Karte seines Vorratsstapels zu spielen verpasst. Und wenn er und sein Gegner ihre Vorratsstapelkarten auf denselben Aufbaustapel spielen wollen, weil dessen Wert am nächsten liegt, schafft A^{Lea} es nicht den Gegner zu blockieren, sondern spielt ihm die – von beiden – benötigten Karten zu.
Doch im Spiel von A^{Lea} gegen zwei A^{B}s sieht man, wie sehr A^{Lea} den Spielerfolg ihres direkt nachfolgenden Gegners beeinträchtigt: Abbildung 5.4 (1). Da jedoch A^{B1} nichts tut, um den ihm nachfolgenden Spieler am Gewinnen zu hindern sondern ihn teilweise durch seine Spielweise sogar unterstützt, erzielt A^{B2} eine höhere Gewinnquote als A^{Lea}.
Bei Abbildung 5.4 (2) sieht man sehr eindeutig, dass A^{Lea1} – weder von A^{B} noch von A^{Lea2} behindert – eine viel höhere Gewinnquote erreicht (41,44%) als A^{Lea2} (27,33%) und A^{B} (31,23%). Die Beeinträchtigung von A^{Lea1} bei A^{Lea2} wirkt jedoch stärker als die von A^{Lea2} bei A^{B}, da A^{L2} durch den Einsatz von A^{Lea1} in seinem Spiel behindert wird und seine Fähigkeiten nicht mehr voll ausnutzen kann.

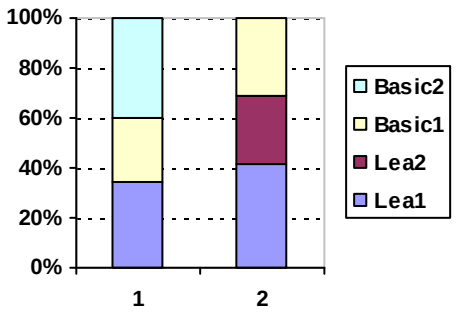

Abb. 5.4: Gewinnstatistik von A^{Lea} im Spiel dreier Agenten. Bei (1) ist Spieler1 A^{Lea}, Spieler2 A^{B1} und Spieler3 A^{B2}. Bei (2) folgen Spieler1 A^{Lea1} Spieler2 A^{Lea2} und Spieler3 A^{B}.

5.5 Gemischte Runde

Bleibt noch zu klären, wie A^M und die zwei gelernten Agenten abschneiden, wenn man sie zusammen in eine Spielrunde setzt, und wer dabei die höchste Gewinnquote erreicht. Wenn man sie in die Reihenfolge ihrer bisherigen Spielergebisse von schlecht nach gut spielen lässt – also A^{TD}, A^{Lea}, A^M, und dann wieder A^{TD} – schneidet A^{Lea} am besten ab, wie auf Abbildung 5.5 zu sehen ist. Setzt man die drei andersherum – A^M, A^{Lea}, A^{TD} – erreicht A^M mit Abstand die höchste Gewinnquote (zu sehen auf Abbildung 5.6), welche die von A^{Lea} beim vorherigen Versuch um über 10 Prozentpunkte übertrifft.
In beiden Fällen schneidet also A^{TD} am schlechtesten ab, und es gewinnt derjenige, der nach A^{TD} – dem schwächsten Spieler – an der Reihe ist. Aber wie hoch dabei die Gewinnquote ist, ist abhängig von den Fähigkeiten des jeweiligen Spielers.

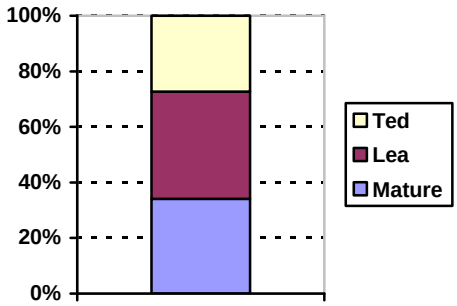

Abb. 5.5: Gewinnstatistik: Spieler 1 ist A^{TD} (27.35%), Spieler 2 A^{Lea} (38,58%) und Spieler 3 A^M (34,07%).

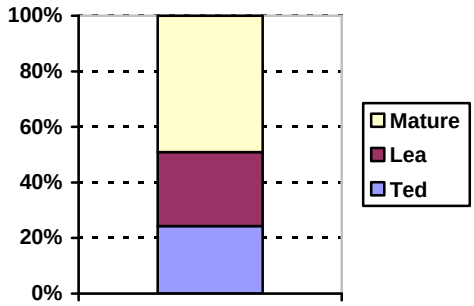

Abb. 5.6: Gewinnstatistik: Spieler 1 A^M (49,04%), Spieler 2 A^{Lea} (26,69%) und Spieler 3 A^{TD} (24,27%).

6 Das Programm – Skip-Bo online

Die in dieser Arbeit vorgestellten Agenten und die Spielumgebung wurden in Java programmiert [10, 11, 12]. Die Spielumgebung existiert in zwei Ausführungen: die Lern-Edition ist speziell zum Lernen der Agenten gedacht, aber auch zu Testzwecken nutzbar, und die Spiel-Edition als Internetanwendung zum Spielen und Testen unter http://www.td-skip-bo.de.vu verfügbar.
Auf der Startseite der Learning Edition, zu sehen auf Abbildung 6.1 (in veränderten Farben), kann ausgewählt werden welche Spiel-Agenten gegeneinander spielen sollen, wobei der dritte Spieler optional ist, und von jedem Agenten-Typ maximal zwei Spieler genommen werden können. Auch die Anzahl der Spielrunden kann hier festgelegt werden. Außerdem gibt es drei verschiedene Spielmodi zur Auswahl: Bei „Lernen“ macht Ted (= A^{TD}) bzw. Lea (= A^{Lea}) nach jedem Spiel ein Update. Bei „L2File“ (= learn to file) wird zusätzlich nach jedem Update eine Testphase zur Bestimmung seiner Qualität eingefügt (d.h. es wird eine bestimmte Anzahl von Spielen gemacht und geschaut, wie viele davon von dem Lernenden gewonnen worden sind) und das Ergebnis zusammen mit den jeweiligen Gewichten in einer Textdatei gespeichert. Bei diesen beiden Varianten kann nur entweder A^{TD} oder A^{Lea} ausgewählt werden, ansonsten wird ein normales Spiel ohne Lerneffekt gestartet. Ein solches Spiel lässt sich auch durch einen Klick auf den „Test“-Button anfangen und ist als Beispiel mit drei Spiel-Agenten auf Abbildung 6.2 zu sehen.
Die Playing Edition verfügt nur über den normalen Spielmodus. Außerdem ist es natürlich auch möglich Mensch gegen Agent zu spielen. Das zugehörige Spielfeld

wurde bereits in Abschnitt 2.1 eingeführt und beispielsweise in Abbildung 2.5 dargestellt. Die komplette Internet-Seite ist auf Abbildung 5.3 zu sehen. Hier kann man nach einem Klick auf den Menüpunkt „Neues Spiel“ links oben unter den vier vorgestellten Skip-Bo-Agenten seinen Gegner aussuchen (oder auch das Spiel nur unter Agenten starten). Durch einen Klick auf den Verdeckten Stapel werden dann die ersten Karten ausgegeben und das Spiel geht los. Zur Hilfe gibt es dort auch Spielanleitungen auf Deutsch und Englisch.
Am Ende des Spiels wird dem (menschlichen) Spieler durch eine kleine Animation angezeigt, ob er gewonnen oder verloren hat: Bei einem Sieg sieht das aus wie auf Abbildung 5.4 abgebildet, eine Niederlage zeigt sich wie auf Abbildung 5.5.

Die Rückmeldungen, die ich durch diese Online-Präsentation bekommen habe, waren durchweg positiv:

- noch vor Fertigstellung aller Spiel-Agenten:
  - „nettes game.... 3 mal gespielt 1 mal gewonnen... leicht zu verstehen...“
  - „tolles spiel, und in der tat leicht zu verstehen und zu lernen... habe gerade 3x in folge gewonnen, und weiter geht’s...“
  - „Skip-Bo kann man spielen ohne aufzuhören, […], es macht Spaß und so was wie dieses Onlinespiel suche ich.“
- nach Fertigstellung, insbesondere von A^M
  - „jetzt kann ich nur noch mit Glück gewinnen, gefällt mir, vorher war’s zu leicht, jetzt find ich es genau richtig.“
  - „Die neue Version ist übrigens toll (spiele schon seit Anfang des Jahres wieder ganz viel).“

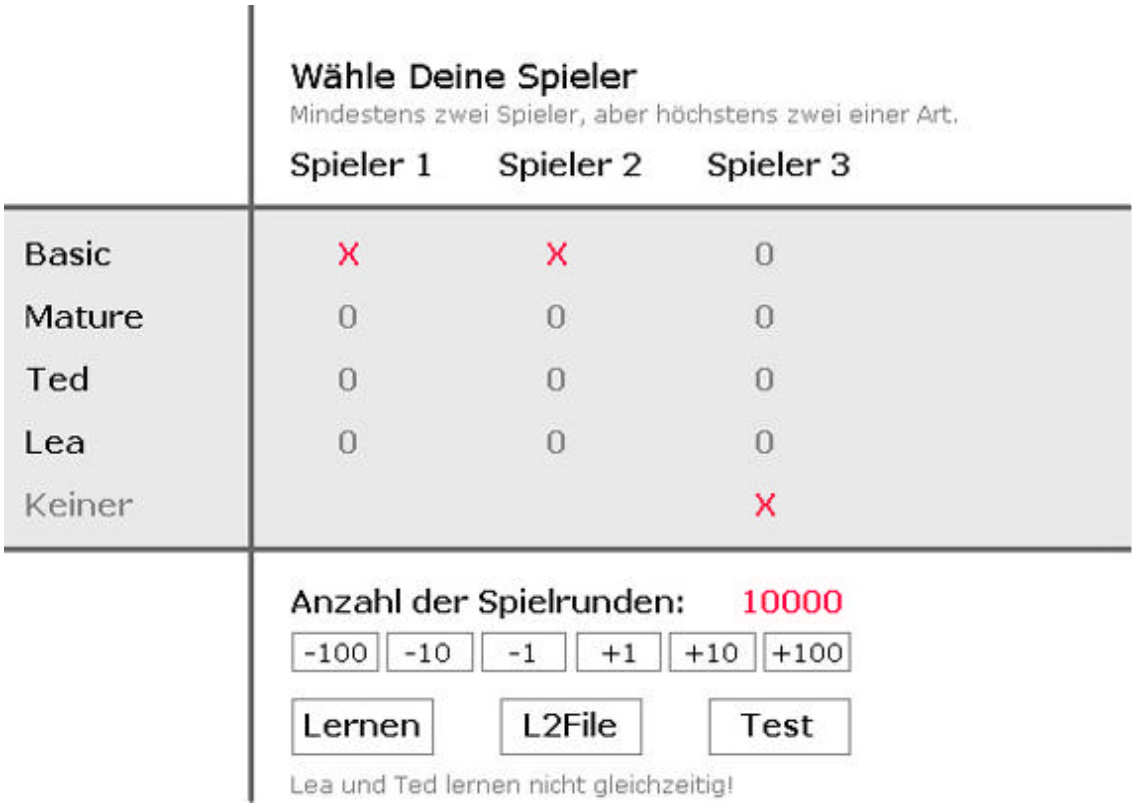

Abb. 6.1: Learning Edition. Startseite zum Auswählen der Spieloptionen.

Abb. 6.2: Learning Edition, Testspiel dreier Agenten.

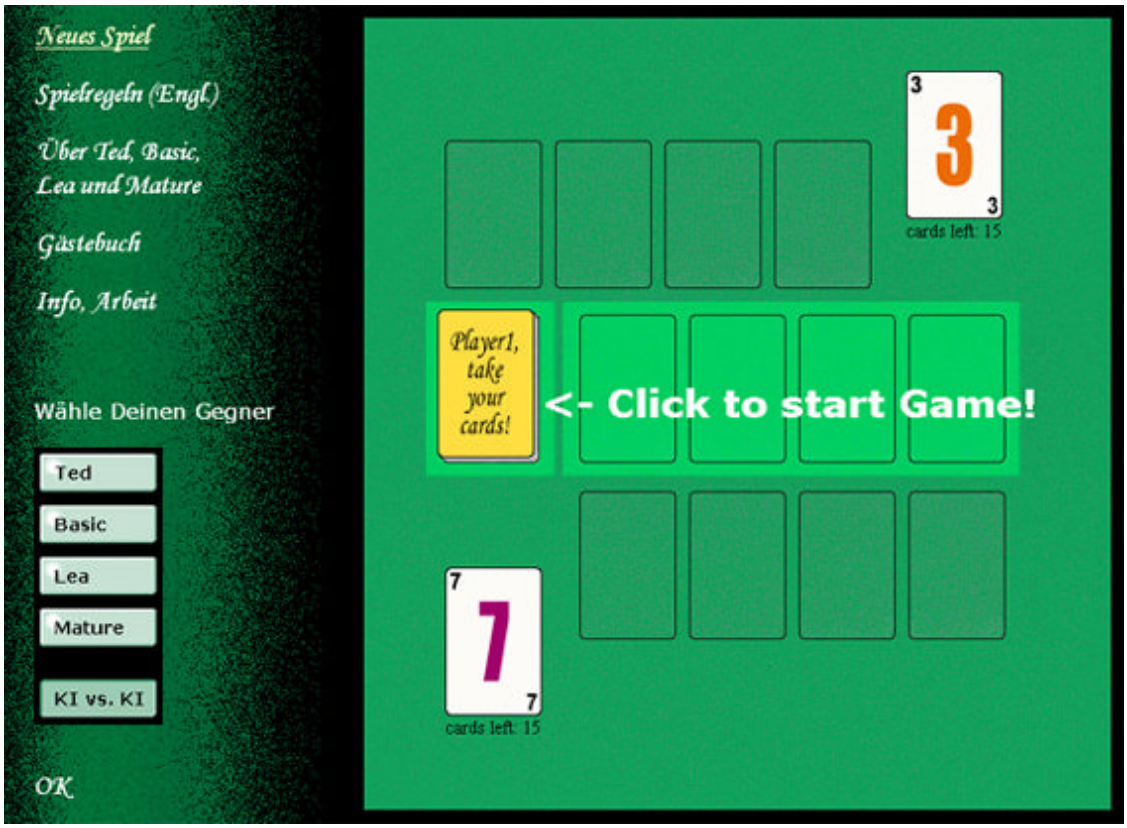

Abb. 6.3: Das Online-Spiel. Die Auswahl des Spielgegners ist unter dem Menüpunkt „Neues Spiel" (rechts oben) und den dadurch links unten erscheinenden Auswahlbuttons möglich. Anschließend wird das Spiel durch einen Klick auf den dann gelb markierten Verdeckten Stapel gestartet.

Abb. 6.4: Gewinner Animation.

Abb. 6.5: Verlierer Animation.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurde das Kartenspiel Skip-Bo vorgestellt, und anschließend der TD(λ)-Algorithmus erklärt. Dieser wurde auf das Spiel angewendet, um neben den zwei regelbasierten Spielern zwei lernende Spiel-Agenten zu erschaffen, die Skip-Bo letztendlich unterschiedlich gut beherrschten.

Für den zweiten lernenden Agenten A^{Lea} wurde hierbei eine Variante des Temporal Difference Algorithmus verwendet, bei dem nicht die verschiedenen Zustände des Spiels bewertet wurden sondern die einzelnen Aktionen, da sich dadurch die Bewertung der Handlungsweise detaillierter darstellen lies.

Dabei stellte sich heraus, dass für das Funktionieren der TD- Methode die Wahl der Parameter, die durch die Gewichte w_i bewertet werden, eine grundlegende Rolle spielt. Der Erfolg ist abhängig davon, wie genau sich die Welt – d.h. das Spiel, seine Zustände oder Aktionen – durch diese Parameter erfassen und bewerten lässt. Ebenso wichtig ist es, die Parameter zu Normieren, damit sie den aktiven Bereich der Bewertungsfunktion nicht verlassen. Die Normierungs-Größe, also der maximale Wert, ist dabei abhängig von Spiel, Agent und gewählten Parametern.

Auch die α- und λ-Parameter müssen individuell an das Spiel angepasst und manchmal auch während dem Lernprozess geändert (verkleinert) werden; unterschiedliche Werte bewirken unterschiedliche Lernweisen und –Erfolge.

Ebenso können auch andere Bewertungsfunktionen verwendet werden. Hier wurde eine hyperbolische Tangentialfunktion benutzt, um den Wertebereich auf das Intervall [-1,1] zu beschränken, aber es ist auch beispielsweise die Anwendung der Exponentialfunktion 1/(1+exp(-x)) und die damit verbundene Einschränkung auf [0,1] denkbar, die in diesem Fall aber nicht so erfolgreich war.

Gestartet werden sollte der Lernprozess mit einheitlichen Gewichten, die dafür nicht unbedingt Null sein müssen, passenden α- und λ-Parametern und im Spiel gegen sich selbst. Sobald der Agent eine gewisse Spielstärke erreicht hat, können die α- und λ-Werte verfeinert und andere Gegner verwendet werden. Die kleineren α- und λ-Werte bewirken eine genauere Anpassung, wobei nicht beide gleichzeitig verändert werden müssen. Und durch die anderen Gegner können weitere Verbesserungen der Spielweise erfolgen, die der Agent im simulierten Spiel gegen sich selbst zumeist nicht erreicht.

Bei Skip-Bo benötigte der TD- basierte Spieler A^{TD} nur 50 Updates, um seine Spielweise zu optimieren, wobei er jedoch nur annähernd die Gewinnquote des regelbasierten A^{B} erzielte. Der Aktions-bewertend lernende Agent A^{Lea} erreichte schon nach weniger als 20 Updates seine Bestform. Diese lag aber trotz allem noch unter den Fähigkeiten des regelbasierten Agenten A^{M}.
Da die bisher erfolgten Anwendungen des TD(λ)- Algorithmus auf Brettspiele wie Schach [6] und Backgammon [2] so erfolgreich waren, wäre es für zukünftige Arbeiten interessant herauszufinden, ob sich bei weiteren Kartenspielen und anderen Anwendungen, bei denen taktische Fähigkeiten gelernt werden müssen, ähnliche Ergebnisse und eine analoge Problematik herausstellen, und ob die Aktions- bewertende TD- Variante dort ebenfalls erfolgreicher ist, als die übliche Zustands- betrachtende.

8 Quellenverzeichnis

[1] Skip-Bo-Spielanleitung der International Games Inc.

[2] TESAURO, Gerald. *Temporal Difference Learning and TD-Gammon*. Online im Internet, URL [cited: 15.10.2006]: http://www.research.ibm.com/massive/tdl.html

[3] BAXTER, TRIDGELL, WEAVER. *Experiments in Parameter Learning Using Temporal Differences*. Online im Internet, URL [cited: 27.08.2006]: http://cs.anu.edu.au/people/Lex.Weaver/pub_sem/publications/ICCA-98_equiv.pdf

[4] SUTTON, BARTO. *Reinforcement Learning: An Introduction*. Online im Internet, URL [cited: 06.09.2006]: http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html

[5] DAUSCHER, UTHMANN. *Kurzskript Reinforcement Learning.* Online im Internet, URL [cited: 06.09.2006]: www.staff.uni-mainz.de/dauscher/softcomp/rlscript.pdf

[6] SAMUEL, A. *Some Studies in Machine Learning Using the Game of Checkers.* IBM Journal of Research and Development.

[7] BAXTER, TRIDGELL, WEAVER. TDLeaf(λ): *Combining Temporal Difference Learning with Game-Tree Search.* Online im Internet, URL [cited: 07.11.2006]: http://cs.anu.edu.au/~Lex.Weaver/pub_sem/publications/ACNN98.pdf

[8] BLOCK, Marco. *Reinforcement Learning goes FUSC#.* Online im Internet, URL [cited: 06.09.2006]: http://page.mi.fu-berlin.de/~fusch/publications/RL-Vortrag_21102003.ppt

[9] MITCHELL, Tom. *Machine Learning*, page 369. Mcgraw-Hill Professional, 1997.

[10] Java.sun.com. Online im Internet, URL [cited: 27.08.2006]: http://java.sun.com/

[11] LEMAY, Laura; PERKINS, Charles L. *Java in 21 Tagen*. Online im Internet, URL [cited: 27.08.2006]: http://jerry.c-lab.de/java/21Tage/httoc.htm

[12] Java Cooperation. Online im Internet, URL [cited: 27.08.2006]: http://javacooperation.gmxhome.de/indexDeu.html