Können Lernalgorithmen interagieren wie im Gehirn?

Bachelor-Thesis von Martin Distler
April 2012

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet für Intelligente Autonome Systeme

Können Lernalgorithmen interagieren wie im Gehirn?

Vorgelegte Bachelor-Thesis von Martin Distler

Betreuung: Prof. Dr. Jan Peters

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 12. April 2012

(Martin Distler)

Inhaltsverzeichnis

1 Einführung

Eine künstliche Intelligenz ist ein von der Menschheit lang gehegter Wunsch, der sich mit dem bronzenen Riesen Talos bis in die griechische Mythologie zurückverfolgen lässt und selbst der bekannte Künstler, Ingenieur und Anatom Leonardo da Vinci bereits im 15. Jahrhundert erstmals Skizzen erstellte, um einen humanoiden Roboter zu erbauen. Ein weiteres solches Beispiel stellt der *Golem* dar, welcher der Sage nach ein von Rabbi Löw aus Lehm erbauter Mensch war, um den in Bedrängnis geratenen Juden Prags zu unterstützen [7].
Durch die Entwicklungen und Forschungen der vergangenen Jahre erscheint eine solche künstliche Intelligenz so nah wie noch nie zuvor: schon heute existieren Maschinen, welche „zu denken" scheinen und Unterstützend im Alltag agieren. Beispiele für solche Maschinen stellen in erster Linie Roboter dar: so agieren Minen-Räum-Roboter oder Mars-Roboter in für Menschen Lebensbedrohlichen Umgebungen, helfen aktiv bei der Aufsuchung verschütteter Menschen [4], oder aber spielen Fußball [5] oder stellen eine Alternative für Haustier-Allergiker dar [1] und es gibt Veranstaltungen, in denen Roboter in den verschiedensten Disziplinen gegeneinander antreten [3]. In naher Zukunft sollen Roboter sogar in der Pflege agieren, um somit in Krankenhäusern und Altenheimen Arbeit abzunehmen, sowie das Leben hilfsbedürftiger Menschen zu Hause zu erleichtern. Erste Erfolge in diesem Bereich erzielte das Georgia Institute of Technology mit seinen Forschungsrobotern Cody und GATSBII, in dem sie hilfsbedürftige Menschen säubern, welche dazu nur noch kaum oder überhaupt nicht mehr im Stande sind [6], [9].
Egal wie unterschiedlich ihre Verwendungszwecke und Arbeitsbereiche sind, so haben sie alle eines gemeinsam: eine Art von *Intelligenz*. Mit ihrer Hilfe finden sie eine Lösung, oftmals sogar die optimale Lösung. Hierbei gibt es zwei grundsätzliche Ansätze, diese Probleme anzugehen: zum einen das *imperative Klassifizieren* der Aufgabenstellung in vom Programmierer / Problemsteller vorgesehene Klassen / Problemfälle um anschließend nach einem festgelegten Ablauf zu agieren, um das Problem zu lösen. Zum anderen gibt es das *eigenständige Klassifizieren*, bei welchem das Problem vom Agenten selbstständig untersucht wird und eine optimale Lösungsstrategie entwickelt wird. Ein wesentlicher Bestandteil des eigenständigen Klassifizieren stellt das Transferieren von Wissen eines Problems auf ein anderes dar. Dabei erscheint die zweite Herangehensweise als die intuitive Bezeichnung der Intelligenz.
Doch eine berechtigte Frage ist: sind diese Roboter „intelligent"? Ein wesentlicher Aspekt der Intelligenz stellt zweifelsohne das *Lernen* dar, doch was ist Lernen *überhaupt* und ist es, um eine künstliche *Intelligenz* zu ermöglichen, auf maschineller Ebene abbildbar? Mit dieser Frage beschäftigt sich die immer wichtiger werdende Disziplin der Informatik *maschinellen Lernens*. In ihr wird versucht das Wissen über das menschliche Lernen auf maschineller Ebene abzubilden, indem aus gesammelten Daten Schlüsse auf Gesetzesmäßigkeiten gezogen werden, um ein gegebenes Problemszenario anhand dieser Schlüsse zu lösen. Doch wie verhalten sich verschiedene Kombinationen der verschiedenen Lernparadigmen in einer Kombination, ähnlich dem menschlichen Lernen? Dieser Frage widmet sich die vorliegende Arbeit.

1.1 Motivation

Die Motivation, das maschinelle Lernen auf das Niveau des menschlichen Lernens zu heben besteht darin, eigenständig agierende und somit vielseitig anwendbare Maschinen zu ermöglichen. Somit könnten Maschinen sich dem Menschen ähnlich fortbilden und sich in verschiedensten Bereichen spezialisieren und Wissen aus anderen Bereichen auf neue Problemszenarien übertragen. Die in den vergangenen Jahren gemachten Entdeckungen in den Bereichen der Neurologie, (Lern-)Psychologie und Informatik ermöglichen es erstmals, die Funktionsweise des Gehirns nachzuvollziehen und zeigen erstmals Parallelen zwischen dem Aufbau und der Funktionsweise des Gehirns und mögliche Abbildung auf maschineller Ebene auf und lässt somit den Schluss realistisch erscheinen, das menschliche Lernen mit Maschinen zu imitieren. Von besonderem Interesse ist hierbei das Zusammenspiel der verschiedenen Lernparadigmen und die daraus resultierenden Möglichkeiten Daten zu verarbeiten.

1.2 Problemszenario

Als Problemszenario wurde ein Labyrinth gewählt in dem sich ein einzelner Agent bewegt (ein sogenanntes *Single-agent system*). Als Ziel des Agenten ist das möglichst schnelle Auffinden des kürzesten Weges vom Startzustand zum Zielzustand definiert. Hierbei stellt das Ziel einen *absorbierenden Zustand* dar: Betritt der Agent dieses Feld ist die aktuelle Spielrunde zu ende und es sind keine weiteren Aktionen möglich bis das Spiel neu gestartet wurde. Zur Lösung dieser Aufgabenstellung werden Lernalgorithmen aus den Bereichen *Reinforcement Learning*, *Unsupervised Learning* und *Supervised Learning* miteinander verbunden, um die Funktionsweise des Gehirns zu imitieren und somit ein möglichst optimales Resultat in den Kriterien *Reduktion der Problemdimension* sowie *Konvergenzgeschwindigkeit* zu erreichen.
Das Labyrinth stellt ein 600 x 300 großes Feld dar, dessen Größe diskretisiert wurde und jedes Objekt eine Größe von 10 x 10 aufweisen. Das ergibt insgesamt eine Problemdimension von 60 x 30 = 1800 Zuständen und ist in x- sowie y-Richtung auf noch größere Problemdimensionen beliebig erweiterbar.

Im Labyrinth herrschen folgende Regeln:

- Jeder Zustand hat die Aktionsmenge $A = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ welche nur durch den Rand des Labyrinths beeinträchtigt ist. Befindet sich der Agent im oberen linken Eck (Position (0, 0)), so ist seine Aktionsmenge somit lediglich $A = \{\downarrow, \rightarrow\}$.
- Der Agent kann durch keine Mauer laufen. Für jeden Versuch in eine Mauer zu laufen, erhält der Agent eine Belohnung von -1000.
- Jeder Zustand hat eine vordefinierte Belohnung von -5, nur der Zielzustand hat eine von 1000.
- Ziel des Agenten ist es einen Weg mit der maximalen Summe aller Belohnungen zu finden, was dem kürzesten Weg des gegebenen Startzustands zum Zielzustand entspricht.

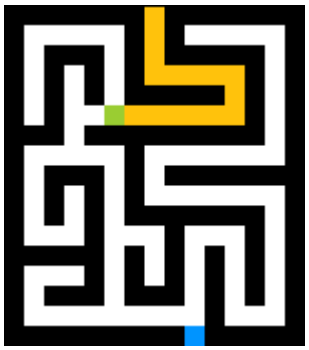

Abbildung 1.1: Beispiel eines verwendeten Labyrinths. Der Agent ist hier grün, das Ziel blau sowie der zurückgelegte Weg gelb eingezeichnet.

1.3 Struktur der Arbeit

Die vorliegende Arbeit beschäftigt sich mit der Idee, die drei Lernparadigmen des Lernens *Reinforcement*, *Supervised* sowie *Unsupervised Learnings* miteinander zu verbinden, um eine mögliche Abbildung des menschlichen Lernens und die daraus resultierenden Ergebnisse für ein gegebenes Problemszenario zu untersuchen. Dabei setzt sich die Struktur der Arbeit wie Folgt zusammen:
Zunächst wird die Frage untersucht, was *Lernen* eigentlich ist und was es ausmacht und es wird versucht die Frage zu beantworten, ob im Kontext der Maschinen das Wort *Lernen* überhaupt angebracht ist. Außerdem werden skizzenhaft die Grundlagen des menschlichen sowie maschinellen Lernens vermittelt und die darin involvierten Elemente benannt, um die Idee der vorliegenden Arbeit herzuleiten. Des Weiteren werden die Gemeinsamkeiten beider Arten zu Lernen betrachtet, um mögliche Architekturen vorzustellen das Problem anzugehen und die Funktionsweise des menschlichen Lernens abzubilden. Anschließend wird in Kapitel 2 ein tieferer Einblick in das maschinelle Lernen geworfen und es wird iterativ die zu untersuchende Abbildung aufgebaut. Hierbei werden die verwendeten Algorithmen samt ihrer Intention beschrieben und ihr theoretischer Nutzen im Problemszenario veranschaulicht. Kapitel 3 beschäftigt sich mit dem verwendeten Setup des Experiments und beschreibt die verwendeten Parameter, die verwendeten Architekturen der einzelnen Kombinationsmöglichkeiten, sowie die untersuchten Labyrinthe selbst. Daraufhin wendet sich Kapitel 4 den mittels diesem Setup erzielten Resultaten und Interpretiert diese. Hierbei ist ein besonderes Augenmerk darauf gelegt, inwiefern sich die einzelnen Lernalgorithmen unterstützen und den Lernprozess somit verbessern. Das letzte Kapitel fasst anschließend die Arbeit in den Punkten *Intention*, *verwendete Algorithmen* und *erzielte Resultate* im gegebenen Problemszenario zusammen. Außerdem wird aus den Resultaten ein Schluss gezogen, inwieweit die Kombination verschiedener Lernparadigmen – dem menschlichen Lernen ähnlich – einen Nutzen erzielt und wie sich die Resultate weiter verbessern ließen.

2 Grundlagen

In diesem Kapitel wird sich mit den Grundlagen des Experiments beschäftigt und es werden die Themen *menschliches* Lernen sowie *maschinelles* Lernen angeschnitten. Hierbei werden die involvierten Elemente genannt und ihre Aufgabe in der Funktionsweise grob analysiert. Das Kapitel beginnt zunächst mit der Erörterung, ob der Begriff des *Lernens* tatsächlich auf Maschinen anwendbar ist.

2.1 Lernen - ein abstrakter Begriff

Ein zentrales Element der Intelligenz stellt zweifelsohne das Lernen dar. Im Rahmen einer künstlichen Intelligenz stellt sich hingegen die berechtigte Frage, ob eine Maschine zu einer solch komplexen und zum teil emotionalen Fähigkeit, wie sie das Lernen darstellt, überhaupt im Stande ist. Um diese leicht philosophische Frage im Ansatz zu beantworten, sei im Folgenden die Definition des Lernens betrachtet sowie zusammengefasst, um daraus zu schließen, ob maschinelles Lernen tatsächlich mit *Lernen* im weiteren Kontext zu vergleichen ist.

> **[Mielke (2001)] [20]** Das Wort „Lernen" geht auf die gotische Bezeichnung für „ich weiß" (*lais*) und das indogermanische Wort für „gehen" (*lis*) zurück *(Wasserzieher, 1974)*. Die Herkunft des Wortes deutet bereits darauf hin, dass Lernen ein Prozess ist, bei dem man einen Weg zurücklegt und dabei zu Wissen gelangt.

Es wird ein Weg zurückgelegt, an dessen Ende zu Wissen gelangt wird. Abstrakt betrachtet ist ein Weg also eine Reihe von Erlebnissen, aus welcher Schlüsse (= *Wissen*) gezogen werden können.

> **[LexiROM (1999)] [15]** [Lernen ist] das Aneignen von Wissen und Kenntnissen bzw. das Einprägen in das Gedächtnis. Das Lernen beinhaltet vor allem auch den Vorgang, im Laufe der Zeit durch Erfahrung, Einsichten oder Ähnlichem zu Einstellungen und Verhaltensweisen zu gelangen [. . .].

Lernen wird in dieser Definition als das Aneignen von Wissen und Kenntnissen, sowie das Einprägen dieser beschrieben, was im Laufe der Zeit in einer Einstellung oder Verhaltensweise resultiert.

> **[Schmitt (1999)] [23]** Unter Lernen verstehen wir den Erwerb, die Veränderung oder den Abbau von Erlebens- und Verhaltensweisen durch bestimmte Umwelterfahrungen.

In dieser Definition kommt eine weitere Komponente hinzu: das *Erwerben* sowie *Verändern* von Verhaltensweisen durch gemachte Erfahrungen.
Zusammenfassend kann man Lernen also als einen Prozess verstehen, bei dem eine Reihe von Erfahrungen erlebt werden und daraus Wissen über sie entsteht. Aus diesem Wissen entsteht eine Verhaltensweise oder Einstellung gegenüber der gemachten Erfahrungen sowie deren Ursprung welche sich durch vergangene oder kommende Erfahrungen verändern kann, sprich die jetzige Vorgehensweise wird evaluiert und den neuen Erfahrungen angepasst um sie zu optimieren.
Dieser komplette Vorgang des Lernens lässt sich zum Beispiel wunderbar bei Kleinkindern mit Greifpuzzles[1] beobachten. Nun versuchen wir uns dies mit Hilfe von 2 Alltagsszenarien zu verdeutlichen und gegenüberzustellen, um zu sehen ob der Begriff des Lernens nun auch tatsächlich auf den Computer übertragbar ist oder nicht.
In diesen Szenarien wird ein Belohnungs-System verwendet um die Beispiele möglichst ersichtlich zu gestalten. Darunter versteht man eine Zuteilung einer Belohnung (oder Bestrafung in Form einer negativen Belohnung) für jede ausgeführte Aktion des zu Lernenden. Ein konkretes Beispiel hierfür wäre ein Leckerli oder aber eine Ermahnung beim Hund um ihn beizubringen was „richtig" oder „falsch" ist. Doch nun zu unseren Beispielen.

[1] Kinderspielzeug welches meist aus Holz ist und das Ziel das korrekte einsetzen der Puzzleteile in die dafür vorgesehenen Löcher ist.

Beispiel: Lernen beim Menschen
Unser erstes Beispiel sei zunächst das menschliche Lernen bei einem Kleinkind und seinem ersten Besuch im Schwimmbad. Hierbei gibt es 3 verschiedene Schwimmbecken: zum einen das Schwimmerbecken, das Nichtschwimmerbecken sowie das Kleinkindbecken. Versucht das Kind nun in eines der Becken zu gehen, in welches es nicht darf, so bekommt es von seinen Eltern Ärger und merkt sich, dass das keine gute Wahl war. Aber da Kinder nun mal Kinder sind und der Forschungsdrang in ihnen brodelt versuchen sie es dennoch mehrere Male bis sie sich merken, dass das zu keinem Erfolg führt und es wird fortan nur noch versuchen in das Kinderbecken zu steigen (oder verbleiben wir der Vorsichtshalber auf „es sollte").
Betrachten wir dieses Beispiel lässt sich schnell die vorher zusammengefasste Definition des Lernens wiedererkennen: das Kind hat eine Reihe von Erfahrungen (jeder einzelne Versuch in ein Becken zu steigen) durchlebt, den Sinn-Zusammenhang zwischen „Ärger bekommen" und „ins falsche Becken gehen" erkannt um anschließend seine Vorgehensweise zu evaluieren und sie anzupassen um immer ins gewollte Nass zu kommen.

Beispiel: Lernen beim Computer
Ein Beispiel zum maschinellen Lernen lässt sich bei den immer beliebter werdenden Staubwisch-Robotern finden. Hierbei sei allerdings nur ein naiver Staubwisch-Roboter als Beispiel genannt, welcher sich durch den Raum navigiert und dabei möglichst viel Staub aufwischt in dem er versucht soviel vom Raum zu wischen, ohne dabei an Gegenständen anzuecken. Weitere Funktionen, wie das abschätzen ob der Raum tatsächlich dreckig ist oder Sensoren welche den Abstand zu einem Objekt messen, seien hierfür irrelevant. Unser Beispielroboter versucht sich also zunächst durch die Wohnung zu navigieren und wird in jeder neuen Umgebung viele male anecken. Jedes mal wenn er aneckt registriert er die relative Position zu seinem Ursprungspunkt und versucht nächstes mal drum herum zu fahren. Nach einigen Durchläufen bildet sich im Roboter nach und nach ein gutes Bild des Raumes ab und er wird in jedem Durchlauf gegen immer weniger Objekte anecken bis er die optimale Vorgehensweise gefunden hat um den Raum zu säubern.
Wenn wir nun auch dieses Beispiel etwas abstrakter betrachten stoßen wir wieder auf unsere Definition des Lernens, denn: der Roboter erlebt eine Reihe von Erfahrungen (jedes anecken sowie jedes nicht-anecken), versucht Sinn-Zusammenhänge zu erkennen („dort ist ein Objekt, darunter kann ich nicht wischen also versuche ich es zu meiden") um anschließend eine möglichst optimale Vorgehensweise zu erkennen um den Raum zu säubern.

Es lässt sich also rein anhand der Definition des Lernens sagen das die Art, wie der Reinigungsroboter und das Kleinkind lernen, zunächst völlig identisch sind. Es wird eine Menge von Erfahrungen gemacht, daraus wird versucht eine Regel oder einen Sinn-Zusammenhang zu erkennen und anschließend werden diese bei der zukünftigen Vorgehensweise mit einbezogen. Folglich ist der Begriff des Lernens auf Mensch und Maschine in seiner Grundessenz gleicherweise anwendbar.

2.2 Das maschinelle Lernen

Die Funktionsweise des maschinellen Lernens beruht auf drei verschiedenen Ansätzen, Wissen zu Gewinnen, zu Strukturieren sowie anschließend zu Verarbeiten. Diese Ansätze, auch *Paradigmen* genannt, sind:

- Reinforcement Learning (*dt. bestärkendes Lernen*)
- Supervised Learning (*dt. überwachtes Lernen*)
- Unsupervised Learning (*dt. unüberwachtes Lernen*)

Jede dieser 3 Paradigmen wird im Folgenden kurz erläutert um einen Einblick zu verschaffen, wie das maschinelle Lernen von statten geht. Dies dient zunächst der reinen Skizzierung der einzelnen Lernparadigmen, da jeder Ansatz seinen eigenen Abschnitt in Kapitel 3 samt Idee, verwendete Algorithmen und Art der Implementierung besitzt.

Reinforcement Learning

Das Konzept des Reinforcement Learnings beruht auf einer Vorgehensweise (*engl. Policy genannt*) die nach und nach zu verbessern gilt um ein konkretes Ziel möglichst optimal zu erreichen. Dabei wird auf Basis der derzeitigen Vorgehensweise eine Aktion ausgewählt welche anschließend bewertet wird. Anhand dieser Bewertung wird die derzeitige Vorgehensweise evaluiert und entsprechend angepasst um anhand mehrerer Iterationen eine möglichst optimale Vorgehensweise zu entwickeln. Sie besteht aus 3 Kernkomponenten sowie in einigen Fällen aus einer optionalen vierten:

Policy π Sie definiert das Verhaltensmuster des Agenten und beschreibt für einen gegebenen Zustand die auszuführende Aktion.

Reward function R Die *reward function* (*dt. Belohnungsfunktion*) beschreibt die für jedes (Zustand, Aktion)-Paar erhaltene Belohnung um somit ein ziel-basiertes Lernen zu ermöglichen.

Value function V Die *value function* (*dt. Werte-Funktion*) beschreibt wie gut eine Vorgehensweise insgesamt ist. Hierbei wird abgeschätzt wie hoch die Summe der zu erwartenden Belohnungen anhand eines gegebenen Startzustands und einer Vorgehensweise ist.

Environment Model F (*optional*) Ein Umgebungsmodell im Sinne des Reinforcement Learnings stellt eine Nachahmung der tatsächlichen Umgebung dar und versucht die Folgezustände und die damit verbundenen Belohnungen anhand eines gegebenen (Zustand, Aktion)-Paares vorherzusagen. Somit dienen Umgebungsmodelle zur Planung und Abschätzung zukünftiger Aktionen und deren Resultate, ohne sie tatsächlich ausführen zu müssen.

Diese Art des Lernens trifft auf die zuvor in Abschnitt 2.1 zusammengefasste Definition des Lernens am offensichtlichsten zu.

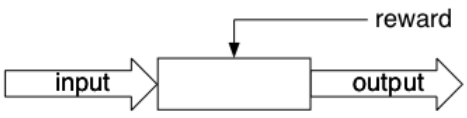

Abbildung 2.1: Reinforcement Learning.

Supervised Learning

Das Ziel des Supervised Learnings besteht darin eine Funktion $f : X \rightarrow Y$ zu finden, welche zu einem gegebenen Eingabewert $x \in X$ einen möglichst genauen Ausgabewert $y \in Y$ liefert. Dabei erhält der Algorithmus eine Menge von Trainingsdaten $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ woraus die zu lernende Funktion f gewonnen wird. Hierbei gilt der Fehler $E = \sum |y_{\text{tatsächlich}} - y_{\text{geschätzt}})|^2$ auf den Trainingsdaten zu minimalisieren wobei $y_{\text{tatsächlich}}$ eine Ausgabe aus einem Trainingsbeispiel $(x_{\text{tatsächlich}}, y_{\text{tatsächlich}})$ darstellt und $y_{\text{geschätzt}}$ anhand der gelernten Funktion zur Eingabe $x_{\text{tatsächlich}}$ erzeugt wird.
Das Supervised Learning findet unter anderem in der Handschrifterkennung Verwendung.
Hierbei erhält der Agent (*der Lerner*, oftmals auch *Agent* genannt) eine Menge von Trainingsdaten in Form von diversen Schriftbildern für Buchstaben und Zahlen. Nach dieser Trainingsphase versucht der Agent nun jeden neu auftretenden Buchstaben oder jede neue Zahl anhand der bisher kennengelernten einzuordnen wobei er den Unterschied in der Schreibweise (hier also der Fehler) zu minimieren versucht. So würde eine unsauber geschriebene 0 mit einer höheren Wahrscheinlichkeit einer sauber 0 zugeschrieben werden, als einer unsauber geschriebenen 8.

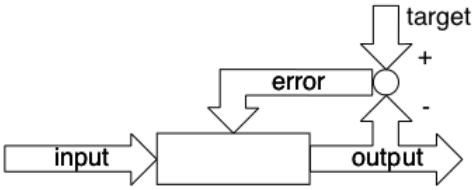

Abbildung 2.2: Supervised Learning.

Unsupervised Learning

Das Ziel des Unsupervised Learning ist das Auffinden versteckter Strukturen innerhalb der gegebenen Eingabedaten mittels Segmentierung[2] (*Clustering*) oder die Reduktion unnötiger Informationen. Hierbei findet keine Evaluierung anhand von Fehlerminimierung (*Supervised Learning*) oder Belohnungen (*Reinforcement Learning*) statt, was sie von den anderen Lernarten unterscheidet.

Abbildung 2.3: Unsupervised Learning.

Um beide Ziele verständlicher zu machen, seien sie in je einem Beispiele erläutert.

Beispiel Segmentierung
Angenommen es gilt ein Puzzel systematisch zu lösen.
Eine Vorgehensweise hierbei wäre zunächst ein beliebiges Puzzel-Stück herauszunehmen und sich die Eigenschaften davon einzuprägen (zum Beispiel ob es eine oder zwei gerade Kanten hat und somit zum Rand gehört, oder aber welche Farben auf dem Puzzel-Stück zu sehen sind). Anschließend legt man es auf einen Haufen.
Nun wird ein zweites Stück genommen und mit den bereits vorher untersuchten Puzzel-Stück anhand der Eigenschaften verglichen. Stimmen die Eigenschaften mit den bisher sortierten Stücken überein so wird es auf den dazugehörigen Stapel gelegt, falls nicht wird es dort abgelegt, wo Platz ist und ist somit der Anfang einer neuen Sortierung. Dies wird nun mit allen Puzzel-Stücken gemacht bis keines mehr übrig ist.
Nun sollten unterschiedliche Sortierungen der Puzzel-Stücke (wie zum Beispiel ein Haufen der alle Eck-Stücke enthält, also solche die 2 gerade Kanten haben) vorliegen. Die Puzzelstücke sind nun *segmentiert* und jeder einzelne Haufen von Puzzel-Stücken stellt einen sogenannten *Cluster* dar.

[2] Segmentierung beschreibt das auffinden von Strukturen in Daten sowie die Einordnung einzelnen Elemente in diese.

Beispiel Datenreduktion
Angenommen ein Konzern möchte wissen, wie seine Kunden gegenüber ihrem Produkt „blauer Hase Schokolade" steht. Hierbei wird eine Stichprobe in einem Einkaufsmarkt erstellt wobei folgende Daten erhoben werden: Name, Alter, Beruf, Familienstatus und Zufriedenheit mit dem Produkt.
Beim auswerten der Daten lässt sich ein Zusammenhang zwischen Alter, Familienstatus und Zufriedenheit feststellen, wobei die Eigenschaften Name und Beruf hingegen keinerlei Einfluss auf die Zufriedenheit haben.
Dies bedeutet, dass die erhobenen Daten der Eigenschaften Name sowie Beruf keinerlei Verwendung im Rahmen der Erhebung haben und somit aus den Datenbeständen gestrichen werden können — die erhobenen Daten wurden auf die Eigenschaften Alter, Familienstatus und Zufriedenheit reduziert.

Da nun die 3 wesentlichen Lernparadigmen des maschinellen Lernens beschrieben wurden und in Kapitel 2 ausführlicher beschrieben werden, widmen wir uns nun dem menschlichen Lernen.

2.3 Das menschliche Lernen

Im menschlichen Lernen sind der derzeitigen Kenntnisse nach im wesentlichen drei Teile des Gehirns involviert: das Kleinhirn, die Großhirnrinde sowie die Basalganglien. Jedes dieser 3 Gehirnareale wird im Folgenden kurz erläutert und es wird versucht die Funktionsweise des jeweiligen Areals zu skizzieren, um einen Einblick in das menschliche Lernen zu erlangen. Beginnen wir mit dem Kleinhirn.

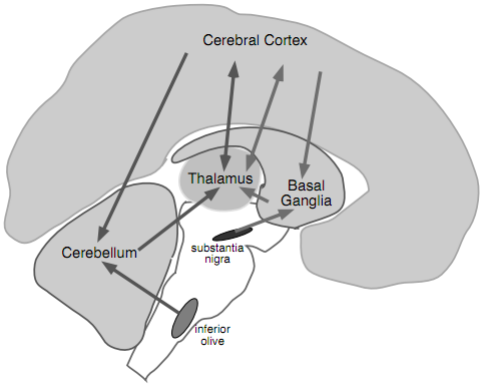

Abbildung 2.4: Seitenansicht des Gehirns und die Verkettung der hier betrachteten Areale.

Kleinhirn *lat. Cerebellum*

Das Kleinhirn, in latein *Cerebellum* genannt, spielt eine wesentliche Rolle in unserer Motorik. Es ist für die Koordination und Feinabstimmung jeder Bewegung zuständig, plant unterbewusst Folgeschritte und erlernt Bewegungsabläufe. Neuerdings geht man sogar davon aus, dass das Kleinhirn kognitive Aufgaben besitzt, wie zum Beispiel Sprachverständnis und Aufmerksamkeit sowie an der Regulierung von Angst- und Lustzuständen teil nimmt [31]. Es gilt allerdings zu unterscheiden, dass das Kleinhirn kein Initiator für Bewegungsabläufe ist, sondern lediglich die Regulierung des eigentlichen Bewegungsablaufes zur Aufgabe hat. Abstrakt betrachtet liegt die Aufgabe des Kleinhirns also vorwiegend darin, für eine initiierte Bewegung (*Eingabe A*) einen möglichst präzisen Ablauf (*Ausgabe B*) zu garantieren. Es versucht also zu jeder Eingabe A eine möglichst optimale Ausgabe B zu haben und sich diese zu merken, sodass man keine Gedanken (*zumindest keine Bewussten*) für routinierte Abläufe, wie zum Beispiel das Laufen oder Greifen nach einem Objekt, aufbringen braucht. Diese Optimierung findet anhand vieler Übungseinheiten statt, bis ein möglichst optimaler Ablauf gefunden wurde der weiterhin verwendet wird.
Ein gutes Beispiel zur Verdeutlichung bietet der bekannte Ausspruch *„Das ist wie Fahrrad fahren, einmal gelernt - nie verlernt!"*. Wenn das Radfahren erlernt wird, wird sich zunächst darauf konzentriert die Beine passend zu bewegen um das Rad überhaupt zu bewegen und dabei – wenn möglich – nicht hinzufallen. Zu Beginn ist es also ein sehr bewusster Vorgang bei dem gelernt wird eine Bewegung zu verinnerlichen. In diesem Prozess evaluiert das Kleinhirn jeden Ablauf und versucht die begangenen Fehler (wie zum Beispiel die zu beginn vernachlässigte Hand-Augen-Koordination) zu beheben und aus ihnen zu lernen um somit einen besseren Ablauf zu finden. Sind diese gefunden so verschwindet die bewusste Konzentration auf diese Bewegungen und das Radfahren wird zu einem natürlichen Prozess wie das Laufen selbst.
Wenn man dieses Verhalten abstrakt betrachtet erschließt sich folgender Gedanke: Das Kleinhirn erhält eine Eingabe *A* („Schwimmen") und versucht dazu eine passende Ausgabe *B* („Arme bewegen") zu finden, welche es anhand eines im Lernprozess auftretenden Fehlers zu optimieren versucht. Anhand dieses Sachverhalts liegt die Vermutung nahe, dass das Kleinhirn auf das zuvor genannte *Supervised Learning* spezialisiert ist. Ob sich diese Vermutung bewahrheitet wird zu einem späteren Zeitpunkt geklärt. Fahren wir zunächst mit den beiden anderen für das Lernen relevanten Gehirnarealen fort.

Großhirnrinde *lat. Cerebral Cortex*

Die Großhirnrinde gilt als das Zentrum aller kognitiven Funktionen, also das Zentrum der Wahrnehmung sowie des Denkens, Sprechens und des Verhaltens. Dabei besitzt es funktionell betrachtet 4 Haupttypen von Feldern welche sich den Aufgaben nach untergliedern lässt: es existiert ein visuelles, ein auditorisches, ein somatosensorisches[3] sowie ein motorisches Feld [8]. In jedes dieser Felder werden die für dieses Feld spezifischen Informationen und Erfahrungen hineinprojiziert. Dabei enthält jedes Feld eine „Karte" welche die räumliche Zuordnung der Information im entsprechenden Feld beschreibt. Neben diesem gezielten projizieren der durch die Wahrnehmung erhaltenen Informationen in die dafür vorgesehenen Bereiche, verarbeitet die Großhirnrinde diese Informationen ebenfalls in ihnen mit Hilfe darauf spezialisierter Areale [11]. Ein gutes Beispiel für ein solches spezialisiertes Areal ist das nach dem deutschen Neurologen Carl Wernicke benannte Wernicke-Zentrum welches zur Aufgabe hat, geschriebene sowie gesprochene Sprache zu verstehen.
Für unseren Zweck, das generelle Betrachten der 3 fürs Lernen wichtigen Gehirnstrukturen reicht es vorerst aus, diesen Sachverhalt zu abstrahieren. Es existiert also zu jeder Art der Wahrnehmung, wie zum Beispiel die Berührung des Rückens, ein spezifisches Feld welches die Informationen gespeichert hat und verarbeitet. Es findet also ein Mapping von Eingabe *A* zu einer Ausgabe *B* statt welches dem des *Unsupervised Learning* entspräche.
Die Funktionsweise lässt sich mit der Analogie einer Telefonzentrale veranschaulichen. Für jedes Anliegen das der Körper hat, wie zum Beispiel „Ich wurde am Bein berührt", leitet die Großhirnrinde diese Informationen an diejenige Stelle weiter, welche auf diese Informationen spezialisiert ist und verarbeitet.

Basalganglien *lat. Basal Ganglia*

Die Funktion der Basalganglien ist bis heute nur im Ansatz verstanden und gilt als äußerst komplex. Es wird ihnen (*hypothetisch*) zugeschrieben für die Selektion von aktuell benötigten Handlungen (motorischen als auch nichtmotorischen) als auch für die Unterdrückung von unerwünschten Handlungen zuständig zu sein. Sie entsprechen somit in gewisser Hinsicht einem „Aktionsfilter" welcher die Menge aller Handlungen von der Großhirnrinde erhält und je nachdem welche Aktionen benötigt werden und welche nicht sie unterdrückt oder aber zur Ausführung auswählt. Dies geschieht durch die Ausschüttung verschiedener Substanzen, so wird unter anderem um eine Aktion anzuregen Dopamin (*als „Glückshormon" bekannt*), ein Neurotransmitter welcher Informationen von Nervenzelle zu Nervenzelle weitergibt, ausgeschüttet, um die Handlung zu initiieren [24]. Forschungen haben hierbei aufgedeckt, dass die Basalganglien besonders im Lernen und Ausführen von zielgerichteten Verhalten eine wesentliche Rolle spielen. So wurde bereits 1993 gezeigt, das die Dopamin Neuronen eines Affens eine gesteigerte Ausschüttung aufzeigten, als das Versuchstier eine unerwartete Belohnung für eine Aktion erhielt oder aber einen Hinweis darauf bekam, dass es in naher Zukunft mit einer Belohnung rechnen konnte [28].
Betrachten wir diesen Sachverhalt abermals von seiner Funktionsweise her, so handelt es sich bei den Basalganglien um ein zielgerichtetes Lern- und Ausführmodul, das je nach Aussicht auf Erfolg gewisse Aktionen fördert und andere hingegen hemmt. Diesen Sachverhalt können wir im *Reinforcement Learning* wiederfinden.

2.4 Verbinden des menschlichen und maschinellen Lernens

Nachdem nun zum einen das maschinelle Lernen sowie zum anderen das menschliche Lernen und die darin involvierten Komponenten betrachtet haben, bleibt nun die Aufgabe übrig, Gemeinsamkeiten zu entdecken. Erste Vermutungen wurden bereits an den jeweiligen Stellen geäußert und werden im Folgenden auf Berufung der Forschungen Kenji Doya et. al. als wahrheitsgetreu betrachtet [10]. Diesen Forschungen zufolge sind die drei zuvor genannten Gehirnareale in der Tat auf die zuvor kennengelernten Lernparadigmen spezialisiert und die sich gegenseitig unterstützende Funktionsweise des Gehirns scheint auf maschineller Ebene abbildbar zu sein. Dabei wurden die einzelnen Areale in ihrer theoretischen Funktionsweise mit der des maschinellen Lernens verglichen, sowie der anatomische Aufbau und die praktische Funktionsweise beobachtet um Schlüsse über auf eine mögliche Implementierung auf dem Computer zu ermöglichen.

Dabei sind im wesentlichen drei Kern-Beobachtungen entstanden welche als Konstruktionsprinzipien der möglichen Architekturen gelten:

1. Die Großhirnrinde arbeitet als Vermittlungsstelle zwischen dem Kleinhirn und den Basalganglien (welche zueinander keine anatomische Verbindung haben) und beinhaltet als solche das Mapping zwischen internem Modell und tatsächlicher Umgebung (*Unsupervised Learning*). Somit entspricht die Großhirnrinde also dem Mapping: internes Modell $\longleftrightarrow$ Umgebung.

[3] Somatosensorisch bedeutet „die Körperwahrnehmung betreffend" und es ist somit die Wahrnehmung durch Haut, Organe, Muskel und Gelenken gemeint [2].

2. Das Kleinhirn enthält ein internes Modell der Umgebung (*Supervised Learning*) und dient somit der Repräsentation der Umgebung mit zusätzlicher Fehlerminimierung.

3. Die Basalganglien werden zur Evaluierung eines gegebenen Zustands benutzt um auf der resultierenden Evaluierung basierend eine Aktion auszusuchen (*Reinforcement Learning*). Sie dienen somit der Aktionsselektion.

Dieser Aufbau der Prinzipien entsteht aus ihren Aufgabenbereichen und lässt sich wie folgt auf das Beispiel *Schwimmen* übertragen.

1. Der Körper nimmt wahr das er sich im Wasser befindet und leitet diese Informationen weiter (Mapping von Umgebung → internes Modell).

2. Die weitergereichten Informationen rufen die im Kleinhirn abgespeicherten Bewegungsabläufe ab (repräsentiert den Status „Im Wasser" mit „Schwimmbewegung").

3. Die Großhirnrinde leitet diese Informationen an die Basalganglien weiter, welche die nun geforderten Aktionen ausführt und unter Umständen andere fürs Schwimmen irrelevante Aktionen hemmt.

Anhand dieser drei Konstruktionsprinzipien wurden folgende Architekturen zusammengefasst, welche versuchen die gegenseitige Unterstützung der zuvor behandelten Gehirnareale abzubilden. Sie wurden zuvor bereits in [10] beschrieben und seien hier erneut benannt, um als Basis des Experiments zu fungieren.

Aktionsselektion

Die Aktionsselektion wird, wie zuvor in Abschnitt 2.3 beschrieben, von den Basalganglien übernommen. Welche möglichen Strategien und somit Architekturen daraus zu schließen sind lassen sich in 2 Kategorien untergliedern. Zum einen in die *reaktive Aktionsselektion (engl. Reactive action selection)* sowie zum anderen in die *vorhersehenden Aktionsselektion (engl. Predictive action selection)*.

Reaktive Aktionsselektion

Die reaktive Aktionsselektion stellt eine einfache *actor-critic* Architektur dar. Hierbei nimmt die Vorgehensweise π die Rolle des *actors* ein, da sie die Aktionen auswählt, und die Value-Function $V(x)$ die des Kritikers, da sie die Vorgehensweise π anhand des zu erwartenden Erfolgs „kritisiert".
Somit führt der Agent zunächst immer die Aktion aus mit der höchsten Gewinnerwartung und evaluiert anhand des erhaltenen Feedbacks (Belohnung r und veränderte Umgebung) seine Verhaltensweise wenn nötig. Diese Aktionsselektion heißt reaktiv, da erst durch den Anstoß einer ausgeführten Aktion eine Evaluierung (und somit Verbesserung) stattfindet.

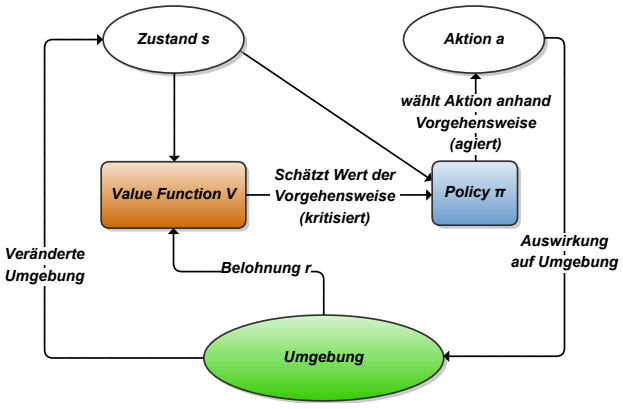

Abbildung 2.5: *Actor-critic* Architektur.

Vorhersehende Aktionsselektion
Bei der vorhersehenden Aktionsselektion wird die Werte-Funktion zusammen mit einem *Umgebungsmodell* verwendet um eine über den gesamten Lernverlauf optimale Vorgehensweise zu entwickeln. Diese Art der Aktionsselektion ist im Gegensatz zur reaktiven Aktionsselektion darauf aus, möglichst frühzeitig eine optimale Vorgehensweise zu entwickeln und lässt sich somit auch als *proaktive Aktionsselektion* bezeichnen.
Da die Auswahl einer Aktion von der Werte-Funktion V abhängt, liegt die vorhersehende Komponente darin diese möglichst frühzeitig anzupassen. Hierbei wird um die Werte-Funktion auf einen Zustand s zum Zeitpunkt t zu schätzen das Optimalitätsprinzip von Bellman[4] angewandt, wodurch die Werte-Funktion somit wie folgt definiert ist:

$$V(s_t) = \max_a(t)[r_{t+1} + \gamma V(s_t)] \tag{2.1}$$

Es wird somit zu diesem Zeitpunkt t diejenige Aktion a ausgewählt, welche die maximale Summe der zu erwartenden Belohnung verspricht. Die zu erwartende Belohnung setzt sich hierbei zum einen aus der *direkt* erhaltenen Belohnung r_{t+1} zusammen, welche die Belohnung des Folgezustands beschreibt, sowie zum anderen aus den *zukünftigen* Belohnungen welche von der Werte-Funktion des aus Aktion a resultierenden Zustands s_{t+1} beschrieben werden. Die zur Berechnung der Werte-Funktionen für die in der Zukunft liegenden Zustände s_{t+1} benötigten Folgezustände werden anhand des Umgebungsmodells $F(s_t, a_t)$ abgeschätzt und liefern somit eine Abschätzung darüber, wie sich zukünftige (Zustand, Aktion)-Paare auf das Lernverhalten auswirken.

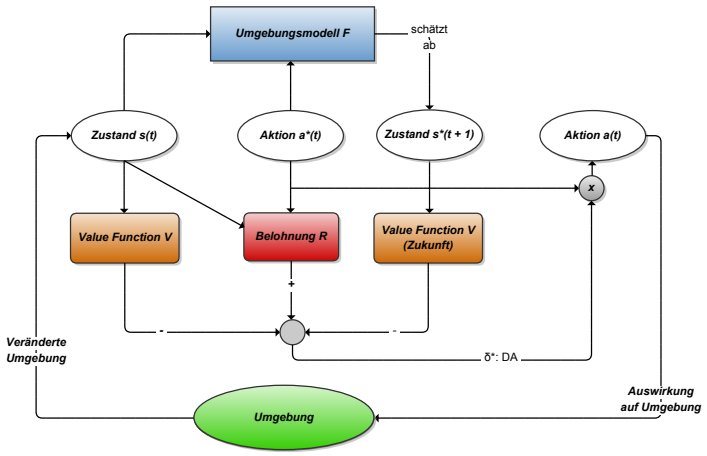

Abbildung 2.6: Vorhersehende Aktionsselektion.

Aktionsselektion mit Supervised Learning

Nachdem nun 2 mögliche Architekturen zur Aktionsselektion vorgestellt wurden, ist nun der nächste Schritt sie anhand der beiden anderen Lernparadigmen zu erweitern. Dieser Abschnitt dreht sich um den möglichen Einsatz des Supervised Learnings bei der Aktionsselektion.

Model basierte Zustandsschätzung
Die bereits vorgestellten Aktionsselektionsmechanismen beruhen auf der Tatsache, das zum Zeitpunkt einer Aktionsselektion die dafür benötigten Daten bereits in einem Zustandsvektor $\mathbf{x}(t)$ vorliegen. Dies kann aber anhand von verspätetem oder aber durch die Sensoren verfälschtes Feedback nicht immer gewährleistet werden, wodurch die Aktionsselektion entweder eine falsche Wahl trifft oder aber zu überhaupt keiner Wahl im Stande ist, da ihr die Informationen fehlen und in den meisten Fällen Zeit ein kritischer Faktor ist. Man stelle sich ein Roboter vor, der gerade einen Schritt ausführt und während der Fuß in der Luft ist keine Folgeaktion für mehrere Millisekunden wählen kann. Das Resultat wäre ein unsauberer Bewegungsablauf wodurch ein Sturz des Roboters wahrscheinlicher wird.
Eine mögliche Unterstützung stellt nun das interne Modell der Umgebung des Supervised und Unsupervised Learnings dar. Anhand dieses Modells könnten alle benötigte Informationen des nun aktuellen Zustands und seiner Folgezustände

[4] Das Optimaltitätsprinzip von Bellman, benannt nach Richard Bellman, besagt, dass sich die optimale Lösung eines Optimierungsproblems (um das es sich hier handelt da es gilt die Werte-Funktion zu optimieren) aus einer Menge von optimalen Teillösungen zusammensetzt.

abgeschätzt werden, wodurch die Aktionsselektion zu jedem Zeitpunkt einen (wenn auch nur geschätzten) Zustandsvektor $\hat{\mathbf{x}}(t)$ hätte, auf dessen Grundlage sie ihre Entscheidungen treffen kann. Hierbei wird verspätetes Feedback durch eine erste Schätzung ausgeglichen und ein verfälschtes Ergebnis kann unter Umständen anhand des Erwartungswerts näher an das tatsächliche Ergebnis approximiert werden. Mögliche Verfahren zur Zustandsschätzung sind zum einen der Kalman Filter [22], zum anderen der Smith Prädiktor [19].
Ein solches Verfahren findet auch im menschlichen Gehirn bei der Objekterkennung durch Somatosensorik statt, bei dem das tatsächliche Gefühl eines Gegenstands mit bisher gemachten (im internen Modell verarbeiteten) Erfahrungen verglichen wird und diesem Ergebnis entsprechend geurteilt wird, ob es sich um den damit assoziierten Gegenstand handelt oder nicht [12]. Auch findet dieses Verfahren des Vergleichs zwischen *Erwartung* und *erhaltene Information* bei der Beurteilung von präzisem Timing bei Bewegungsabläufen statt [14].

Simulation anhand eines Umgebungsmodells
In dem zuvor beschriebenen Aktionsselektionsmechanismus *Vorhersehende Aktionsselektion* wird mit Hilfe des internen Umgebungsmodells zu einem Zeitpunkt t diejenige Aktion a_t gewählt, welche die maximale Summe der zu erwartenden Belohnungen verspricht. Eine mögliche Verbesserung stellt das Simulieren einer *Schrittfolge* anhand dieses internen Umgebungsmodells dar, bei dem nicht nur *eine* Aktion und ihre Auswirkung, sondern eine *Sequenz* von Aktionen samt derer Auswirkungen geplant wird. Eine solche Erweiterung ist allerdings nur dann möglich, wenn der Faktor Zeit kein kritischer und die Menge der möglichen Aktionen klein ist. Ist dies der Fall können mehrere zum Zeitpunkt t mögliche Szenarien simuliert und entsprechend evaluiert werden (*„offline learning" genannt*). Somit können die Werte-Funktion V und Vorgehensweise π den simulierten Erfahrungen entsprechend angepasst werden, ohne eine große Menge an tatsächlich erlebten Erfahrungen zu benötigen. Der Erfolg dieses Vorgehens hängt dabei wesentlich von der Genauigkeit des internen Umgebungsmodells ab.

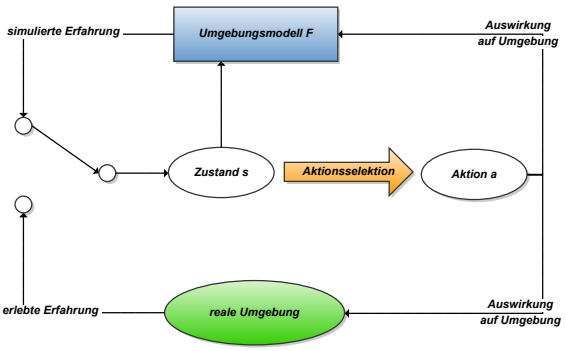

Abbildung 2.7: Simulation von Erfahrungen anhand der Nutzung des Umgebungsmodells *F*.

Verkapseln erlernter Vorgehensweisen
In vielen Systemen stellt der Faktor Zeit eine kritische Komponente dar, wie zum Beispiel in der Evaluation des nächsten Schrittes im Rahmen einer Bewegungssequenz. Die bisher vorgestellten Architekturen weisen allerdings aufgrund ihrer Kommunikation zwischen internem Modell und Aktionsselektion des Agenten einen nicht geringen Mehraufwand in den beiden Punkten Rechenzeit sowie Speicherbedarf auf, was dazu führen kann, dass sie in zeitkritischen Bereichen ungenügend sind. Eine mögliche Verbesserung für ein solches zeitkritisches Szenario stellt das *Verkapseln erlernter Vorgehensweisen* dar. Hierbei werden die oben genannten Architekturen zur reinen Lernphase verwendet und sobald die Veränderungen in der Vorgehensweise π in einem Zustand s unter einer definierten Schranke fällt, wird die zuletzt erlernte Regel dieses Zustands als Standard definiert und nicht weiter verfeinert. Somit fallen im Verlauf der Zeit zunehmend mehr Lernschritte weg wodurch zum einen Rechenzeit und zum anderen der damit verbundene Speicherplatz verringert wird. Ist anschließend eine Vorgehensweise π soweit verfeinert das alle Zustände eine Standardaktion definiert haben, kann sie als einzelne Vorgehensweise verkapselt abgespeichert werden, was weitere Rechenschritte obsolet macht. Außerdem wird durch das verkapselte Speichern (alle Regeln geordnet und zusammenhängend in einem Speicherbereich) eine verbesserte Zugriffszeit ermöglicht, wodurch die Zeit, welche zum Treffen einer Entscheidung benötigt wird, weitgehend minimalisiert wird und nur noch von der Zugriffszeit abhängt. Dies könnte besonders in Systemen von Nutzen sein, welche mehrere Aufgabenstellungen parallel lernen. Ist eine Vorgehensweise zu einer Aufgabenstellung in ausreichendem Maße erlernt, stehen ab diesem Zeitpunkt mehr Rechenkapazitäten für die anderen Aufgaben bereit.

3 Auswahl der Algorithmen

Dieser Abschnitt wendet sich im Folgenden den drei verwendeten Paradigmen *Reinforcement*, *Unsupervised* sowie *Supervised Learning* zu, indem ihre Anwendung auf das Problemszenario verdeutlicht und ihre zu Grunde liegenden Intuition und Verbindung zum menschlichen Lernen erläutert wird. Des Weiteren werden Einblicke in die daraus resultierenden Implementierungen und Architekturen verschafft, sowie ein erster Blick auf die zu erwartenden Resultate geworfen. Sei zunächst mit der Basis zur Lösung des Problemszenarios begonnen: dem Auffinden eines Weges mittels Reinforcement Learning.

3.1 Verwendete Notationen

In den kommenden Abschnitten sind Folgende Notationen verwendet worden.

t = Diskreter Zeitschritt
s_t = Zustand zum Zeitpunkt t
a_t = Aktion zum Zeitpunkt t
(s_t, a_t) = (Zustand, Aktion)-Paar π = Vorgehensweise (*Policy*), Entscheidet welche Aktion gewählt wird
α = Lernfaktor, beschreibt wie viel der Agent aus jeder einzelnen Aktion lernt
γ = Discount-Factor[1], beschreibt die Tendenz nach zukünftigen Belohnungen zu suchen
Q = Quality / Qualität des zugehörigen (Zustand, Aktion)-Paars
R = Reward / Belohnung des zugehörigen Zustands
$r = R(s_t)$, die Belohnung des Zustands s_t

Oftmals wird hingegen die Benennung des diskreten Zeitschritts t ausgelassen, sodass beispielsweise s geschrieben, s_t aber gemeint ist.

3.2 Reinforcement Learning

Das Reinforcement Learning wurde vom Behaviorismus inspiriert und basiert auf dem Grundgedanken, die Vorgehensweise von Mensch und Tier in den Naturwissenschaften abzubilden. Hierbei wird versucht die Eigenschaft zu imitieren, aus Erfahrungen Schlüsse zu ziehen und diese in die kommenden Entscheidungen einfließen zu lassen. Hierbei bewegt sich ein Reinforcement Learning Agent durch eine bisher unbekannte Umgebung und versucht seine Aktionen, welche mit einer Belohnung versehen sind, basierend auf zuvor getroffenen Entscheidungen so zu wählen, das die Summe der Belohnungen maximiert wird. Somit stellt das Reinforcement Learning eine Ansammlung von Methoden dar, welche die optimale Lösung eines sequentiellen Entscheidungsproblems (ein sogenanntes *Markov-Entscheidungsproblem*, engl. *Markov-Decision-Process* kurz MDP genannt) approximiert[2] [26].
Als Lernalgorithmen in der Sparte Reinforcement Learning wurden zunächst die Algorithmen *SARSA* sowie *Q-Learning* untersucht. Beide Lernalgorithmen stellen den Standard im Bereich Reinforcement Learning dar und sind auf das Problemszenario ideal anwendbar, da es sich um ein diskretisierbares Problemszenario handelt. Am Ende dieses Abschnitts wird aus den zwei vorgestellten Algorithmen einer ausgewählt, welcher die Grundlage der weiteren Kombinationen mit Unsupervised und Supervised Learning Algorithmen bildet.

[1] Je kleiner der Discount-Factor γ ist, desto mehr wird der Agent die Aktion a_t in Zustand s_t wählen, welche zum Zeitpunkt t die höchste *direkte* Belohnung verspricht.

[2] Approximation ist allgemein in Synonym für Annäherung, ist in der Mathematik allerdings präziser formuliert.

3.2.1 Grundlagen

Wie zuvor in der Einleitung dieses Abschnitts beschrieben, beschäftigt sich das Reinforcement Learning mit der Approximation einer optimalen Lösung im Rahmen eines sequentiellen Entscheidungsproblems, in dem die Auswirkungen getroffener Entscheidungen dazu verwendet werden eine Vorgehensweise π zu entwickeln. Da lediglich die eigenen Entscheidungen in die Approximation einfließen, ist kein „Lehrer" von Nöten, welche die getroffenen Entscheidungen bewertet, wodurch es sich vom Supervised Learning unterscheidet. Hierbei interagiert ein *Agent* mit der Umgebung auf einer diskreten Zeitskala $t = 0, 1, 2, 3, ...$ in den damit verbundenen Zuständen s_t, indem eine Aktion a_t anhand der derzeitigen Vorgehensweise π gewählt wird und der Agent somit in Zustand s_{t+1} endet, wobei er eine Belohnung r_{t+1} für dieses Paar (s_t, a_t) erhält. Anhand dieser Belohnung versucht der Agent nun seine Vorgehensweise so anzupassen, sodass er die Summe der zu *erwartenden* Belohnungen $E = \sum_{i=0}^{\infty} \gamma^i r_{t+i+1}$ zu maximieren versucht. Hierbei gibt es zwei grundsätzliche Möglichkeiten, die Vorgehensweise optimieren: zum einen lässt sich eine optimale Zustandsfunktion V^*, welche versucht, jedem Zustand s die maximale Summe der zu erwartenden Belohnungen zuzuweisen (*value-mapping*). Anschließend versucht man anhand dieser Funktion diejenigen Folgezustände zu wählen, welche die höchste zu erwartende Summe verspricht und folgt somit diesen Zuständen zum Ziel. In manchen Szenarien ist es allerdings hingegen für den Agenten nicht möglich eine konkrete Aussage zu erteilen, mit welcher Aktion er in den Folgezustand gelangt, welcher die höchste Summe aufweist. Somit gibt es als zweite Möglichkeit das sogenannte *action-value-Mapping* bei dem eine optimale Aktion-Zustands Funktion Q^* gelernt wird, welche jedem (Zustand, Aktion)-Paar (s, a) einen Wert zuweist, sodass gilt: In Zustand s wird mittels Aktion a der Wert $Q^*(s, a) = \max_{\pi} E\{r_{t+1} + \gamma * r_{t+2} + ... | s_t = s, a_t = a\}$ erreicht. Da es im Rahmen des Szenarios intuitiver ist die getroffenen Aktionen in den dazugehörigen Zuständen zu beurteilen, wurde im Folgenden ein solches *action-value-Mapping* gewählt.

3.2.2 Aktionsselektion

Einer jeden Vorgehensweise π liegt ein Selektionsmechanismus zu Grunde, welcher die nächste Aktion basierend auf den derzeitigen Informationen (*den Q-Werten*) auswählt. In der hier verwendeten Implementierung wurde zur Selektion eine sogenannte *Greedy* Vorgehensweise gewählt welche im Folgenden kurz erläutert wird.

Greedy (*dt. „gierig"*)
Unter einer Greedy-Vorgehensweise versteht man, das immer diejenige Aktion gewählt wird, welche den momentan größten Nutzen bringt. Hat der Reinforcement Agent die Wahl zwischen einer Aktion, welche eine Wertung in Höhe von 10 aufweist und einer zweiten Aktion mit einer Wertung in Höhe von 11, so wird die letztere gewählt, da sie zu diesem Zeitschritt t den momentan höchsten Nutzen erzielt.
Dadurch entsteht allerdings ein Problem: durch das kontinuierliche wählen der *momentan* besten Aktion konvergiert das Verfahren zu einem *lokalem Maximum* ohne in Betracht zu ziehen, dass das lokale nicht dem globalen entspricht und findet somit *eine* Lösung aber nicht unbedingt *die beste* Lösung. Das Problem ist an folgendem Graph verdeutlicht: Der Agent würde hier gegen das lokale Maximum zu seiner linken konvergieren und hätte somit *eine* Lösung gefunden, welche zum Ziel führt. Allerdings sieht man anhand des globalen Maximums, dass diese Lösung nicht die *optimale* sein kann – was allerdings das eigentliche Ziel des Agenten darstellt.

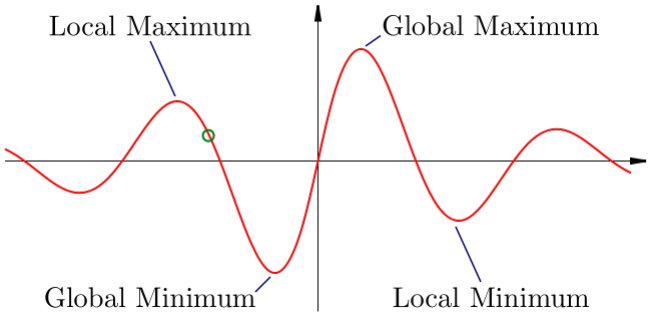

Abbildung 3.1: Darstellung des Problems: Position des Agenten in grün.

Um diesem Problem entgegen zu wirken wurde das sogenannte ϵ-Greedy Verfahren gewählt.

ϵ-Greedy
Hierbei handelt es sich um ein Greedy-Verfahren welches zu einer Wahrscheinlichkeit $p = 1 - \epsilon$ die Aktion mit dem höchsten Q-Wert wählt und zu einer Restwahrscheinlichkeit von $p = \frac{\epsilon}{N-1}$ (mit $N =$ Anzahl möglicher Aktionen) eine zufällige nicht-optimale Aktion der Aktionsmenge. Je höher der ϵ-Wert gewählt ist, desto höher ist die Exploration des Labyrinths und es wird somit zunehmend wahrscheinlicher, dass der optimale Weg gefunden wird.

3.2.3 Q-Learning

Einer der zwei hier vorgestellten Algorithmen stellt das *ein-Schritt Q-Learning* dar [29]. Bei diesem Algorithmus wird die Q-Learning ist ein Reinforcement Learning Algorithmus welcher versucht für jede Aktion a eines Zustands s mittels Exploration den Wert $Q(s,a)$ und somit die Vorgehensweise π zu verfeinern. Hierbei wird eine Wertefunktion Q^* gelernt, welche versucht die Summe der zu erwartenden Belohnungen abzuschätzen [30]. Dies geschieht in dem der Agent versucht für jeden möglichen Zustand s die jeweils optimale Aktion a zu finden, auch wenn diese nicht der Vorgehensweise entspricht (*off-policy* Verfahren genannt). Dies geschieht indem zur Berechnung des Q-Werts des aktuellen (Zustand, Aktion)-Paars derjenige Q-Wert des (Zustand, Aktion)-Paars verwendet wird, welcher die maximale Summe der Belohnungen verspricht.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{alter Wert}} + \alpha * [\overbrace{R(s_{t+1}) + \gamma \underbrace{\max_{a_{t+1}} Q(s_{t+1},a_{t+1})}_{\text{max. Wert des Folgezustands}}}^{\text{gelernter Wert}} - \underbrace{Q(s_t,a_t)}_{\text{alter Wert}}]$$

Formel 3.1. Update-Regel des Q-Learning-Verfahrens.

Dadurch erlernt der Q-Learning Agent nicht nur die optimalen Aktionen der Zustände, welche durch das Folgen der Vorgehensweise besucht werden würden, sondern auch diejenigen um die besuchten Zustände herum. Somit bildet sich in der Q-Werte Tabelle eine Struktur ab, welche neben der zu verfeinernden Vorgehensweise π weitere mögliche Vorgehensweisen π_n aufweist, da mehr exploriert wird. Eine wichtige Eigenschaft des Q-Learnings, die Konvergenz gegen die optimale Lösung, wurde zuvor von Watkings und Dayan im Jahre 1992 bewiesen, wodurch das Auffinden einer optimalen Lösung im Folgenden gewährleistet ist.

Listing 3.1: Q-Learning Algorithmus

```
1 Initialisiere Q(s, a) (s ∈ Zustand, a ∈ Aktionen in s)
2 Wiederhole für jede Episode
3  Initialisiere s
4  Wiederhole für jeden Schritt der Episode (while s != end)
5      Wähle a von s anhand der Policy (eps-Greedy)
6      Führe a aus, betrachte r und s'
7      Q(s, a) = Q(s, a) + alpha * [r + gamma * maxQ(s', a') - Q(s, a)]
8      s = s'
```

3.2.4 SARSA

SARSA[3] ist ein Algorithmus der Reinforcement Learning Familie, welcher versucht eine Vorgehensweise π anhand der *bisher ausgeführten* Aktionen iterativ zu verfeinern (so genannte *on-policy Verfahren*). Hierbei wird strikt der gegebenen Vorgehensweise π gefolgt um die Q-Werte der (Zustand, Aktion)-Paare zu erlernen indem die Q-Werte des Folgepaares, welche ebenfalls anhand der Vorgehensweise berechnet werden, mit ein berechnet werden. Durch dieses kontinuierliche mit einfließen lassen des Folgeschritts und dessen Q-Werts, werden die Q-Werte der zu der Vorgehensweise entsprechenden (Zustand, Aktion)-Paare iterativ verfeinert.
Der Unterschied zum zuvor beschriebenen Q-Learning steckt im Detail: während bei SARSA zur Berechnung des aktuellen $Q(s,a)$-Werts der Folgeschritt mit eingerechnet wird ($Q(s_{t+1}, a_{t+1})$), wird beim Q-Learning hingegen derjenige Schritt mit eingerechnet, welcher den maximalen Q-Wert der möglichen (Zustand, Aktion)-Paare verspricht.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{alter Wert}} + \alpha * [\overbrace{R(s_{t+1}) + \gamma \underbrace{Q(s_{t+1},a_{t+1})}_{\text{Wert des Folgezustands}}}^{\text{gelernter Wert}} - \underbrace{Q(s_t,a_t)}_{\text{alter Wert}}]$$

Formel 3.2. Update-Regel des SARSA-Verfahrens.

Das hat zur Folge, dass SARSA diejenigen $Q(s,a)$-Werte erlernt, welche durch das Verwenden der Vorgehensweise π aufkommen und somit mehr der Exploitation (*Ausnutzen des Wissens*) als der Exploration (*Erkundung der Umgebung*) angehört. Q-Learning hingegen beruht durch die Hinzunahme des (Zustand, Aktion)-Paars, welches den maximalen Q-Wert der Folgezustände besitzt, mehr auf der Exploration als auf der Exploitation und lernt somit die Q-Werte der Vorgehensweise π anhand einer mehr explorierenden Vorgehensweise. Ein Resultat hieraus ist, dass schlechte Erfahrungen wie bspw. das laufen gegen eine Mauer oder in einen Abgrund auf SARSA einen wesentlich negativeren Einfluss auf die Vorgehensweise hat, als es bei Q-Learning der Fall wäre und SARSA somit eine *sichere* Vorgehensweise entwickelt welche Gefahren zu meiden versucht als die von Q-Learning entwickelte. Mehr zu dieser Beobachtung im Abschnitt *Q-Learning oder SARSA?* auf Seite 20. Das ϵ wird im verwendeten Algorithmus aus Konvergenzgründen einem Vorschlag nach [26] über den Verlauf der Lernzeit verringert und beginnt bei einem Startwert von 1, sodass gilt: $\epsilon = \frac{1}{t}$. Damit wird SARSA im Verlauf des Lernens gieriger und führt ab genügend großem t keine Zufallsaktionen mehr aus.

Listing 3.2: SARSA Algorithmus

```
1  Initialisiere Q(s, a) (s ∈ Zustand, a ∈ Aktionen in s)
2  Wiederhole für jede Episode
3   Initialisiere s
4   Wähle a von s anhand der Policy (eps-Greedy)
5   Wiederhole für jeden Schritt der Episode (while s != end)
6       Führe a aus, betrachte r und s'
7       Wähle a' von s' anhand der Policy (eps-Greedy)
8       Q(s, a) = Q(s, a) + alpha * [r + gamma * Q(s', a') - Q(s, a)]
9       s = s'
10      a = a'
```

[3] Der Name leitet sich von der Berechnung des neuen Qualität-Werts ab. Hierzu wird der jetzige Zustand s_t, die ausgeführte Aktion a_t, die daraus resultierende Belohnung (Reward) r sowie der Folgezustand s_{t+1} und die darauf folgende Aktion a_{t+1} benötigt. Als Tupel geschrieben also $(s_t, a_t, r, s_{t+1}, a_{t+1})$.

3.2.5 Beispielanwendung

Beide Verfahren wurden im Folgenden auf zwei der Labyrinthe des Experiments angewendet. Zunächst sei bemerkt, dass sowohl Q-Learning als auch SARSA in beiden Fällen den Erwartungen gemäß einen optimalen Weg vom Startzustand zum Zielzustand gefunden haben. Neben jedem Labyrinth stellen außerdem Diagramme die Performanz der beiden Algorithmen dar. Hierbei entspricht die X-Achse der Anzahl der untersuchten Episoden[4], die Y-Achse beschreibt hingegen zum einen die Summe der durchschnittlich erhaltenen Belohnungen und andererseits die Anzahl der durchschnittlich durchgeführten Schritte während der zugehörigen Episode.

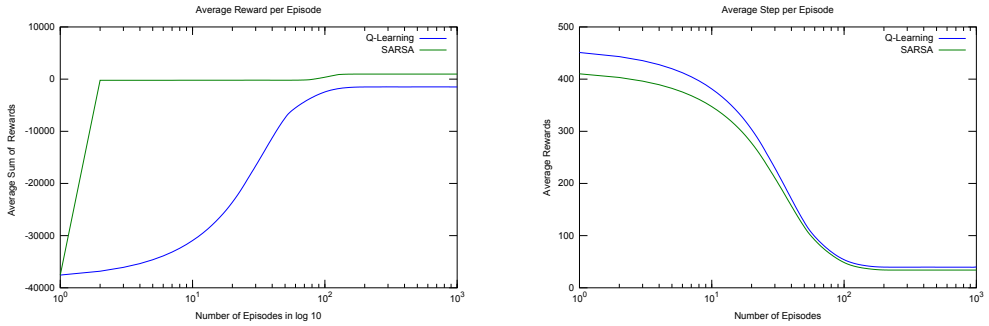

Abbildung 3.2: Q-Learning und SARSA auf das *klassische* Labyrinth angewendet, α = 0.5 γ = 0.98, ϵ = 0.1 über 1000 Episoden mit x-Skala in log_{10}.

Hierbei ist zu sehen, dass SARSA aufgrund seiner gieriger werdenden Aktionsselektion bereits ab der zweiten Episode dem Greedy-Verfahren gleicht und sich dem optimalen Weg rasant annähert. Das Q-Learning hingegen exploriert das Labyrinth ausgiebig und trifft aufgrund des ϵ-Greedy Verfahrens oftmals falsche Entscheidungen, welche drastische Auswirkungen auf das Labyrinth haben. Nichts desto trotz ähnelt sich die Anzahl der durchschnittlich durchgeführten Schritte ca. ab der 50. Episode und lediglich die vom ϵ-Greedy verursachten Fehlschritte unterscheiden beide Verfahren.

3.2.6 Q-Learning oder SARSA?

Nach einer Erläuterung der Algorithmen und einer Beispielanwendung auf das gegebene Problemszenario stellt sich nun die Frage, mit welchem der beiden Verfahren weitergearbeitet wird. Hierzu sei ins Gedächtnis gerufen, dass das Ziel des Agenten das Auffinden des kürzesten Weges von seiner Startposition aus zur Zielposition darstellt und somit ein Ausgleich zwischen Exploration und Exploitation gefunden werden muss. Dieser Ausgleich findet bei einem Off-Policy Verfahren, wie es Q-Learning ist, deutlich mehr statt, da es primär nicht auf die ausgeführte Aktion achtet als vielmehr auf die generell sich bildende Struktur der Q-Werte und lernt somit auch andere Vorgehensweisen als die derzeitig verwendete. SARSA hingegen arbeitet daran iterativ seine Policy zu verbessern (daher auch *On-Policy*) in dem es zukünftige Aktionen gemäß der Vorgehensweise mit in Erwägung zieht. Daraus resultiert das schlechte Erfahrungen (dargestellt durch (Zustand, Aktion)-Paare, welche den Agenten in eine Mauer laufen lassen) dazu führen können, dass weitere Explorationsschritte ausbleiben da sie unter Umständen eine schlechte Belohnung erwartet. Somit lernt SARSA im Vergleich zum Q-Learning *sicherer*, in dem es versucht schlechten Erfahrungen aus dem Weg zu gehen, beim Q-Learning hingegen wird dies in Kauf genommen um ein besseres Mapping von Q-Werten auf die einzelnen (Zustand, Aktion)-Paare zu erhalten.
Dieses Verhalten lässt sich besonders gut im sogenannten „*Cliff walking*"-Szenario beobachten. Hierbei handelt es sich um das in Abbildung 3.3 zu sehende Level. S beschreibt die Startposition des Agenten, G hingegen das Ziel. Jede Aktion hat eine Standardbelohnung von $r = -1$, allerdings haben Aktionen welche in einem der mit *The Cliff* gekennzeichneten Felder enden, eine Belohnung von $r = -100$. Außerdem wird der Agent zurück zur Startposition geschickt sobald er durch eine solche Aktion auf ein Cliff-Feld gelangt.

[4] Eine *Episode* entspricht das einmalige Auffinden eine beliebigen Weges vom Start- zum Zielzustand

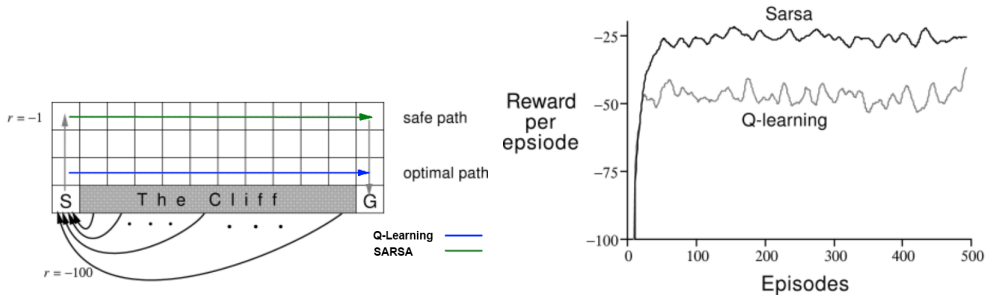

Abbildung 3.3: Unterschied von Q-Learning und SARSA anhand des *cliff walking*-Tests.

Hierbei lässt sich deutlich der Unterschied zwischen *On-Policy* und *Off-Policy* erkennen: der On-Policy Agent findet einen sicheren Pfad bei dem er nicht Gefahr läuft in ein Cliff-Feld zu gelangen, hat allerdings nicht den kürzesten Weg gefunden. Der Q-Learning Agent hingegen hat den Optimalen Pfad entdeckt, obwohl seine Leistung im Graph daneben merkbar schlechter ausfällt als die des SARSA Agenten. Die einfache Erklärung hierfür liegt in der Exploration des Agenten im Zusammenhang mit dem ϵ-Greedy Verfahren, wodurch – trotz richtiger Policy – es gelegentlich vorkommt, dass der Agent in ein Cliff-Feld gelangt.

Als Fazit daraus ziehen wir zweierlei: zum einen ist es einem Off-Policy Verfahren eher möglich in „gefährlichen Regionen" (wie es in dem hier verwendeten Problemszenario in Form von engen Gängen der Fall ist) eine optimale Vorgehensweise zu finden. Da Labyrinthe in ihrer Natur zumeist verwinkelt und eng sind würde ein Off-Policy Verfahren somit bessere Resultate erzielen. Zum anderen lässt Q-Learning mehr Exploration zu was in späteren Phasen dieser Arbeit von Vorteil sein wird, da somit wesentlich mehr Informationen aus dem Labyrinth über seine Struktur zu erlangen sind, welche später in den Bereichen Unsupervised sowie Supervised Learning benötigt werden. Aus diesen beiden Gründen bildet im Folgenden das Q-Learning den verwendeten Algorithmus im Bereich Reinforcement-Learning.

3.3 Unsupervised Learning

Wie zuvor in der Einleitung erwähnt behandelt das Unsupervised Learning das Problem, versteckte Strukturen in Eingabedaten wiederzufinden oder die mit den Daten verbundene Problemdimension zu verringern. Dabei findet keine Fehlerkorrektur statt sondern es werden Kriterien festgelegt anhand dessen die Daten einer Struktur zugeordnet oder reduziert werden.
Die Intention des Unsupervised Learnings im Rahmen des gegebenen Problemszenarios besteht darin, die Komplexität des Labyrinths auf markante (Zustand, Aktion)-Paare zu reduzieren und somit potenzielle Fehlentscheidungen zu verhindern was den Lernprozess beschleunigt. Hierzu wurde die folgende Methode entworfen um das Clustering von Gängen innerhalb des Labyrinths zu ermöglichen.

3.3.1 Herleitung der Idee

Begeht man ein Labyrinth so ist intuitiv klar, wann es in Frage kommt die Richtung zu wechseln – nämlich genau dann, wenn man an einer Kreuzung oder Ecke vorbei kommt, denn erst dann besteht überhapt die Möglichkeit von *gehe nach Norden* zu *gehe nach Süden*, *gehe nach Westen* oder *gehe nach Osten* zu wechseln. Befindet man sich inmitten eines Ganges, in dem es nur vor oder zurück geht, so sollte man in der Regel nicht auf den Gedanken kommen gegen eine Wand laufen zu wollen und der einzige Richtungswechsel welcher in Frage kommt ist das Inverse zur jetzigen, das *zurück gehen*. Für einen Reinforcement Learning Agenten ist das hingegen nicht so leicht, schließlich ist er ja gewillt das Labyrinth zu erkunden und hat zu Beginn keine Ahnung wohin ihn ein (Status, Aktion)-Paar bringen wird und versucht entsprechend öfters mit dem Kopf durch die Wand zu gehen.
Betrachtet man ein Labyrinth hingegen von oben so wird man sich deren möglicherweise vorhandenen Gänge und dessen Kreuzungen und Ecken bewusst und kann schnell entscheiden an welchen Ecken eine Entscheidung möglich ist, denn abstrakt betrachtet stellt ein Labyrinth einen gerichteten Graphen dar, wobei die Zustände die Knoten und die Aktionen die Kanten zu den Folgeknoten darstellen. Betrachten wir nun also Abbildung 3.4 welches oben ein Labyrinth samt Übergänge darstellt sowie darunter den dazugehörigen Zustandsübergangsgraph des Labyrinths aufweist.
Hinweis: Hierbei sei allerdings angemerkt das Aktionen, welche in einem nicht definierten Zustand oder einer Mauer landen, zur besseren Lesbarkeit nicht eingezeichnet wurden – sie sind aber nichts desto trotz vorhanden und stellen eine Schleife zum jeweiligen Zustand dar. So ergäbe zum Beispiel (S_0, ↑) als Folgezustand wieder S_0.

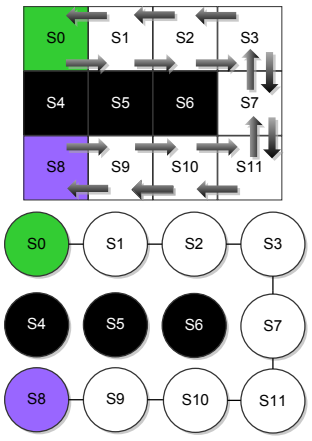

Abbildung 3.4: Labyrinth und dazu gehöriger Zustandsübergangsgraph. Spieler in, Ziel in lila.

Um also von Grün zu Lila zu gelangen ist die Menge der Aktionen der Reihe nach: $\{\rightarrow, \rightarrow, \rightarrow, \downarrow, \downarrow, \leftarrow, \leftarrow, \leftarrow\}$. Eine wichtige Beobachtung hierbei ist das wir in Zustand S_3 und Zustand S_{11} einen Richtungswechsel vornehmen können und in diesem Fall sogar müssen. Diese besonderen Knoten sind im Folgenden als *Transit(-knoten)* bezeichnet. Ein Transitknoten ist hierbei als ein Knoten definiert, bei dem ein Richtungswechsel von *horizontal* zu *vertikal* (oder umgedreht) im Verlauf des Spielverlaufs stattgefunden hat. Des Weiteren wird der Start- sowie Endknoten auch als Transit definiert. Die Intuition besagt nun, dass in der obigen Aktionsmenge die Aktionsfolge $\{\rightarrow, \rightarrow, \rightarrow\}$ zu einer einzelnen Aktion $\rightarrow$ zusammengefasst werden kann, da die Aktion $\rightarrow$ genau so lange Sinn macht, bis wir einen

Transitknoten erreichen, an dem es die Wahl gibt, eine andere Richtung einschlagen zu können (analog für die beiden Teilmengen $\{\downarrow, \downarrow\}$ sowie $\{\leftarrow, \leftarrow, \leftarrow\}$). Es macht also Sinn sich zu merken, welche Knoten einen Transitknoten darstellen.

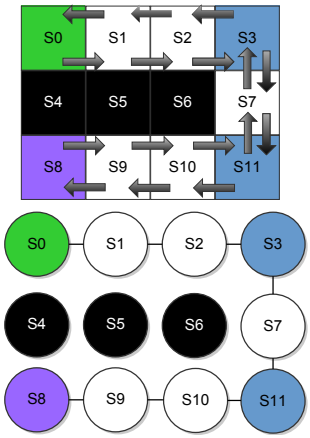

Abbildung 3.5: Finden und markieren der Transitknoten. Transitknoten in blau.

Abbildung 3.5 stellt nun das Labyrinth und dessen Graphen dar, nachdem die Transitknoten gefunden und markiert wurden. Anhand dieser Markierung ist es nun möglich eine Regel aufzustellen, die besagt: *Ändere deine jetzige Aktion nur dann, wenn du auf einem Transitknoten endest*. Diese Regel wirkt intuitiv auf den Menschen, ist aber von wesentlicher Bedeutung für den Agenten der auf dem Labyrinth agiert, da nun mit einem Mal ein Großteil der „negativen Aktionen" (welche im gleichen Zustand landen und somit keinen Nutzen haben, wie zum Beispiel gegen eine Mauer zu laufen) wegfallen. Dadurch hat der Agent nur noch die Aktionen $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$ in den Zuständen $\{S_0, S_3, S_8, S_1 1\}$, in den Zuständen $\{S_1, S_2, S_4, S_5, S_9, S_{10}\}$ hingegen nur noch die Aktionen $\{\leftarrow, \rightarrow\}$. Was wir also mit der Markierung der Transitknoten und der oben aufgestellten Regel implizit *meinen* ist in Abbildung 3.6 zu sehen – eine *Reduktion auf das Wesentliche*, kurz: auf die Transitknoten.

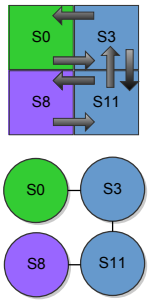

Abbildung 3.6: Reduktion der Zustandsmenge auf zur Lösung relevante Zustände.

Anhand dieser Konzentration auf die Transitknoten ist es nun möglich sogenannte *Makro-Aktionen* zu definieren. Diese Makro-Aktionen bestehen aus einer Sequenz von Aktionen, verändern dabei allerdings nicht ihre Richtung. Ziel der zuvor beschriebenen Reduktion ist nun solche Makro-Aktionen zu definieren, sodass es stets einen direkten Weg zwischen zwei Transitknoten existiert, den der Agent wählen kann. So besteht nun die Möglichkeit von S_0 direkt nach S_3 zu wandern, was effektiv die Zustände $\{S_1, S_2, S_7, S_9, S_{10}\}$ weggefallen lässt, in denen der Agent eine „negative Aktion" ausführen könnte, da der Agent, solange er eine Makro-Aktion ausführt, keine Wahl hat, außer ihr zu folgen. Es gibt somit in diesem Beispiel nur noch 4 relevante Zustände, in denen der Agent eine wichtige Entscheidung zu treffen hat, was das Lernen merkbar beschleunigen sollte.

3.3.2 Makro-Aktionen im Entscheidungsproblem

Wie zuvor bereits beschrieben stellen Makro-Aktionen eine *Sequenz von Aktionen* dar, bei dessen Ausführung der Agent keine Wahlmöglichkeiten in den Zwischenzuständen treffen kann. Diese Makro-Aktionen und der somit resultierende Graph aus Abbildung 3.6 stellen ein sogenanntes semi-Makrov-Entscheidungsproblem (*engl. semi-Markov decision process, kurz SMDP*) dar.
Der Unterschied lässt sich wie folgt verdeutlichen: Angenommen man möchte eine Tür öffnen. Im Rahmen eines MDPs würde diese Aufgabe die einzelnen Schritte *gehe zur Tür*, *greife nach Türknauf*, *drehe Türknauf* und *öffne Tür* beinhaltet, welche jeweils einen diskreten Zeitschritt benötigen würden. Diese Aufgabe lässt sich allerdings abstrahieren zu einem SMDP, bei dem nur noch der Schritt *öffne Tür* existiert, welcher intern die zuvor genannte Folge von Schritten ausführt. Dieses Schema wird in diesem Algorithmus ausgenutzt, um zunächst das MDP zu einem SMDP zu transformieren (siehe Grafiken 3.4, 3.5 sowie 3.6) und anschließend mit diesen Makro-Aktionen, welche aus dem SMDP extrahiert werden, das MDP zu erweitern, was den Lernvorgang somit beschleunigen sollte [27].

Da nun die Aktionsmenge des Agenten mit Makro-Aktionen erweitert wurde und das zuvor beschriebene Q-Learning auf Basis einzelner Aktionen arbeitet, muss das verwendete Q-Learning Verfahren an die Makro-Aktionen angepasst werden. Hierzu wird die gelernte Funktion Q^* mit weiteren Einträgen erweitert, sodass sie zum einen die Aktionen a als auch die Makro-Aktionen m eines Zustands s (*anhand der Summe der zu erwartenden Belohnungen*) bewerten kann. Da Makro-Aktionen, im Gegensatz zu simplen Aktionen, mehrere Schritte enthalten und somit eine andere Belohnung für die Ausführung erhält, ist eine Veränderung der zuvor in Gleichung 3.1 kennengelernten Update-Regel nötig. Dazu wird die Update-Regel für einen Zustand s_t und eine Makro-Aktion m wie folgt verändert:

$$Q(s_t, m) \leftarrow \underbrace{Q(s_t, m)}_{\text{alter Wert}} + \alpha * [\overbrace{r + \gamma^n \underbrace{\max_a Q(s_{t+n}, a)}_{\text{max. Wert des Folgezustands}}}^{\text{gelernter Wert}} - \underbrace{Q(s_t, m)}_{\text{alter Wert}}]$$

Formel 3.3. Update-Regel für Makro-Aktionen.

Hierbei entspricht n der Anzahl der Zeitschritte, welche benötigt werden die gegebene Makro-Aktion m auszuführen und Zustand s_{t+n} ist somit derjenige Zustand, in welchen der Agent endet wenn er in Zustand s_t die Makro-Aktion m ausführt. Da der Agent während einer Makro-Aktion n-Aktionen ausführt (da jede einzelne Aktion einen Zeitschritt benötigt), erhält er ebenfalls n-Belohnungen. Da diese Belohnungen allerdings in der Zukunft liegen, muss jede Belohnung ihrer Entfernung (*in Zeitschritten gemessen*) nach mit dem Discounting-Factor γ gewichtet werden. So ergibt sich für den in Gleichung 3.3 genannten Wert r folgende Berechnung:

$$r = \gamma^0 r_{t+1} + \gamma^1 r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} = \sum_{i=0}^{n-1} \gamma^i r_{t+1+i}$$

Die daraus resultierende Erweiterung des Q-Learning Verfahrens ist das *Makro Q-Learning* [18] und verwendet für normale Aktionen die Update-Regel aus Gleichung 3.1 und für Marko-Aktionen die Update-Regel aus Gleichung 3.3.

3.3.3 Algorithmus

Der zur obigen Verhaltensweise passende Algorithmus besteht im Kern aus 3 Elementen:

1. Exploriere das Labyrinth (*Explorationsphase*)
2. Finde und markiere Transitknoten
3. Fasse Transitknoten zusammen

Explorationsphase
Die Explorationsphase ist ein zentrales Element des Verfahrens um das Zusammenfassen von Gängen und somit ihrer Transitknoten überhaupt zu ermöglichen. In ihr erkundet der zu Beginn eines jeden Labyrinths blinde Agent das Labyrinth mit Hilfe seiner (Zustand, Aktion)-Paare und baut nach und nach einen Übergangsgraphen (wie er im obigen Beispiel zu sehen ist) auf und entwickelt somit über die Zeit gesehen ein Abbild seiner Umgebung. Erst wenn dieser Graph zu einem gewissen Grad aufgebaut wurde, ist es sinnvoll Gänge zusammenzufassen und dessen Transitknoten miteinander zu verbinden. Geschieht dies zu früh besteht die Gefahr für die optimale Lösung wichtige Transitknoten zu überspringen und somit nicht zu entdecken. Ein guter Zeitpunkt die Explorationsphase zu beenden ist beispielsweise das Ende der ersten Lernepisode, da nachdem das Ziel einmal betreten wurde alle zur Lösung relevanten Kreuzungen und Ecken begangen wurden und an ihnen somit die Chance besteht in eine andere Richtung fortzusetzen.

Finden und markieren der Transitknoten
Das Finden und Markieren der Transitknoten findet parallel zur Explorationsphase statt. Jedes neue vom Agenten gefundene (Zustand, Aktion)-Paar wird mit Hilfe seiner Aktion in insgesamt 3 Hauptcluster zusammengefasst. Ist die Aktion ↑ oder ↓ wird der Zustand dem *vertikalen Cluster* zugeordnet, ist sie ← oder → hingegen dem *horizontalen Cluster*. Aus der obigen Definition der Transitknoten geht nun hervor das der dritte Cluster, der *Transitcluster*, die Schnittmenge des horizontalen und vertikalen Clusters ist und enthält somit genau diese Zustände in denen eine horizontale sowie vertikale Aktion stattgefunden hatte.

Zusammenfassen der Transitknoten
Nachdem die Explorationsphase beendet wurde, ist es nun möglich die gefundenen Transitknoten zusammenzufassen. Hierbei wird der Übergangsgraph erweitert, in dem zur bereits bestehenden Aktionsmenge Makro-Aktionen hinzukommen, welche Transitknoten miteinander verbindet. Hierbei findet ein weiteres Clustering statt, bei dem alle Zustände, welche zusammenhängend auf einer vertikalen oder horizontalen Linie liegen, zusammengefasst werden. Anschließend werden alle Transitknoten innerhalb dieser Subcluster miteinander verbunden und es entsteht durch die Erweiterung der Aktionsmengen mit Makro-Aktionen ein Übergangsgraph wie er zuvor in Abbildung 3.6 zu sehen war.

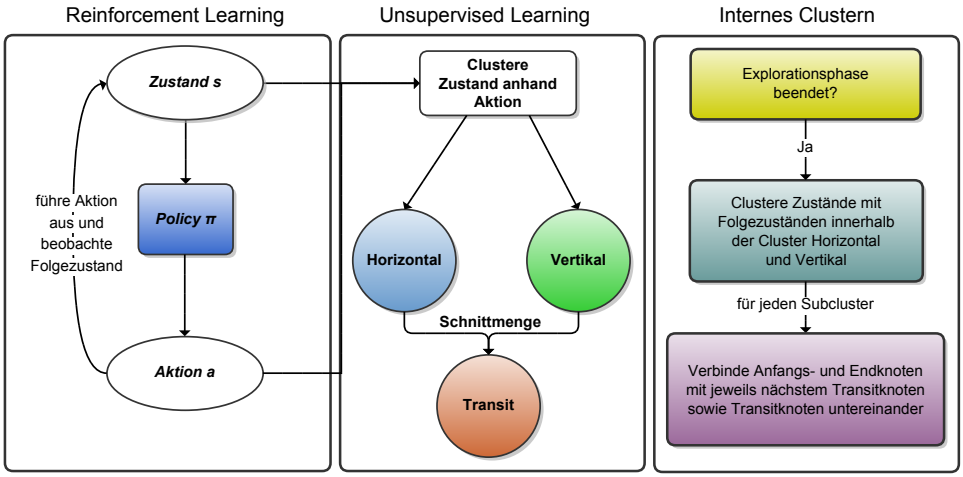

Abbildung 3.7: Visualisierung des Algorithmus und seiner Verarbeitungsschritte.

In Abbildung 3.7 ist der komplette Ablauf nochmal bildlich dargestellt. Der Reinforcement Learning Agent agiert auf einem Zustand s_t und führt die anhand der Vorgehensweise π gewählte Aktion a_t aus und gelangt somit in Zustand s_{t+1}. Jedes (s_t, a_t)-Paar wird an das Unsupervised Learning übermittelt und der Zustand s_t entsprechend seiner Aktion a_t den beiden Clustern *Horizontal* und *Vertikal* zugeordnet. Anschließend wird der Schnitt beider Cluster ermittelt welcher die Menge der Transitknoten darstellt. Ist die Explorationsphase zu Ende, startet das interne Clustern bei dem alle zusammenhängende Zustände zusammengefasst und die Transitknoten miteinander verbunden werden.

3.3.4 Beispielanwendung

Das zuvor beschriebene Verfahren wurde nun beispielhaft auf ein Labyrinth des Experiments angewendet, um die Funktionsweise zu demonstrieren. Hierbei stellt Abbildung 3.8 das verwendete Labyrinth, mit den bereits geclusterten Zuständen farbig in Form von kleinen Quadraten markiert, zum einen nach Beendigung der Explorationsphase und zum anderen nach der Beendigung der Lernphase dar. Hierbei sind grüne Punkte Elemente des vertikalen Clusters, lila Punkte des horizontalen Clusters und blaue Punkte stellen Elemente des Transitclusters dar.

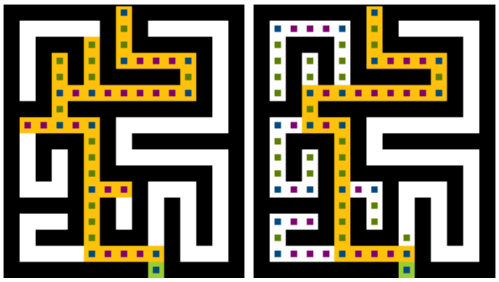

Abbildung 3.8: Das *klassische* Labyrinth nach Beendigung der Explorationsphase (links) und am Ende der Lernphase mit eingezeichneten Zuständen der drei Cluster (rechts).

Es ist zu sehen, dass die benötigte Strecke vom Start- zum Zielzustand in diesem Falle aus nur 12 Transitknoten (inklusive Start- und Zielzustand) besteht und somit nur diese vom Reinforcement Learning untersucht werden brauchen. Dies entspricht in diesem einfachen Beispiel allein einer Reduktion von 35 auf 12 Zuständen und sollte somit das Lernen zum einen zeitlich beschleunigen (*gemessen an benötigter Zeit in ms*), als auch das Lernverhalten allgemein verbessern.

3.4 Supervised Learning

In den zuvor vorgestellten Algorithmen des Reinforcement sowie Unsupervised Learnings ging es darum, zunächst eine Lösung in Form eines Weges von *Start* nach *Ziel* zu finden sowie die gesamte Problemdimension in Form eines Zustandsübergangsgraphen auszudrücken und mittels zusammenfassen unnötiger Zustände zu verringern. Somit sind auch zwei für den Menschen zunächst intuitive Teilprobleme des Labyrinth-Problems gelöst, nämlich zum einen das generelle Auffinden eines Weges von *Start* zu *Ziel* sowie das Folgen eines Ganges in einer Richtung, da sonst keine andere Wahl bleibt. Im Rahmen des Labyrinth-Problems gibt es allerdings ein drittes Teilproblem, genauer ein *Entscheidungsproblem*, welches zunächst ebenfalls intuitiv wirkt: das Entscheiden ob eine Aktion a in Zustand s überhaupt Sinn ergibt. Konkret sind hierbei Aktionen gemeint, welche den Agenten gegen die Wand laufen lassen.
Um dies entscheiden zu können wird eine Funktion f benötigt, welche für ein gegebenes (Zustand, Aktion)-Paar als Eingabe eine Aussage trifft, ob diese Aktion *gültig* oder *ungültig* ist. Dies ist, wie zuvor in Abschnitt 2.1 beschrieben, die Aufgabe des *Supervised Learnings*.

3.4.1 Grundsätzliche Idee

Um diese zur Klassifizierung benötigte Funktion f zu finden, wird zunächst eine Menge von Trainingsdaten benötigt aus welcher f abgeleitet werden kann. Im Falle des Anwendungsszenarios besteht die Menge der Trainingsdaten somit aus Tupeln der Art $((s,a),v)$, wobei s ein Zustand, a eine Aktion des Zustands s und $v \in \{gültig, ungültig\}$ eine Aussage über die Gültigkeit des (Zustand, Aktion)-Paars (s,a) ist.
Um diese Trainingsdaten zu erhalten lassen sich im Folgenden 2 Ansätze verfolgen: zum einen das *on-the-fly*-Verfahren, zum anderen das *precomputed*-Verfahren. Beide werden im Folgenden kurz erläutert.

On-the-fly-Verfahren
Beim *on-the-fly*-Verfahren agiert der Supervised Learning Agent im Hintergrund des Reinforcement Agenten und erzeugt die Trainingsdaten anhand der durch das Reinforcement Learning entstehenden Informationen iterativ. Für jede ausgeführte Aktion a in Zustand s gibt es einen Folgezustand s' anhand dessen der Supervised Agent entscheiden kann, ob eine Aktion als gültig oder ungültig zu klassifizieren ist. Ist der Folgezustand s' gleich dem Ausgangszustand s, also $s = s'$, so hat diese Aktion anhand der im Labyrinth geltenden Regeln den Agenten gegen eine Wand laufen lassen und die Aktion a kann somit als ungültig klassifiziert werden. Die Klassifizierungsfunktion ist also für einen Zustand s und seinem Folgezustand s' mittels Aktion a definiert durch:

$$valid(s,a,s') = \begin{cases} s = s' & \text{(s, a) ist ungültig} \\ s \neq s' & \text{(s, a) ist gültig} \end{cases} \tag{3.4}$$

Mittels dieser Funktion und der Informationen des Reinforcement Agenten entwickelt sich durch jeden Schritt des Reinforcement Agenten ein Umgebungsmodell, welches für die bereits gesehenen (Zustand, Aktion)-Paare eine Aussage über die Gültigkeit des Paares treffen kann. Diese Trainingsdaten bilden somit das Umgebungsmodell, welches im Verlauf des Lernens iterativ erweitert wird.

Precomputed-Verfahren
Die Idee des *precomputed*-Verfahrens (zu deutsch Sinngemäß *„vorausberechnendes"* oder *„voraus kalkulierendes"* Verfahren) besteht darin, *vor* der Ausführung des Reinforcement Learning Agenten den Supervised Learning Agenten durch das Labyrinth zu bewegen und in jedem Zustand s alle Aktionen a auszuführen, welche in der Aktionsmenge A_s des jeweiligen Zustands definiert sind, um anschließend den Folgezustand zu betrachten. Das *precomputed*-Verfahren verwendet ebenfalls die in Gleichung 3.4 definierte Klassifizierungsfunktion. Anhand dieser Informationen bildet das Supervised Learning ebenfalls ein *Umgebungsmodell*, welches für jedes (Zustand, Aktion)-Paar eine Aussage $v \in \{gültig, ungültig\}$ aufweist.
Eine wichtige Eigenschaft dieses Verfahrens ist, dass das Umgebungsmodell *vollständig* ist, da in *jedem* möglichen Zustand *jede* Aktion ausgeführt wurde.

Beide Ansätze erzeugen somit einen Trainingsdatensatz, welcher als Umgebungsmodell fungiert. Nun gilt es aus diesem Trainingsdatensatz eine Funktion $f : S\ x\ A_s \rightarrow \{gültig, ungültig\}$ mit $S =$ Menge aller Zustände und $A_s =$ Aktionsmenge in Zustand s zu finden, welche zu einem gegebenem (Zustand, Aktion)-Paar eine Aussage über ihre Gültigkeit trifft. Um diese Funktion zu finden lassen sich außerdem folgende Eigenschaften festhalten:
Da das Labyrinth im Problemszenario statisch ist und sich somit im Laufe der Zeit nicht verändert, verändern sich auch die internen Aussagen des Umgebungsmodells nicht. Daraus folgt, dass wenn ein (Zustand, Aktion)-Paar irgendwann $gültig$ war, so ist es auch weiterhin $gültig$. Gleiches gilt für $ungültige$ (Zustand, Aktion)-Paar. Außerdem können über (Zustand, Aktion)-Paare, welche nicht in den Trainingsdaten enthalten sind, keine Aussage getroffen werden. Damit eine Aussage getroffen werden könnte, müsste Aktion a in Zustand s ausgeführt worden sein, um den Folgezustand zu ermitteln und das (s, a)-Paar somit auf Gültigkeit zu prüfen. Da es keinen konkreten Zusammenhang zwischen den einzelnen Zuständen und Mauern gibt und somit Mauern zufällig verteilt sein könnten, kann keine Folgerung auf die Gültigkeit von nicht gesehenen (Zustand, Aktion)-Paaren anhand von Umgebungsinformationen ermittelt werden.
Aus diesen Beobachtungen lässt sich also schließen das eine Fehlerminimierung nicht nötig ist, da der Trainingsdatensatz (und somit das Umgebungsmodell) immer die einzig korrekte Antwort enthält. Außerdem lässt sich keine Abschätzung über noch nicht gesehene (Zustand, Aktion)-Paare treffen. Es liegt also der Schluss nahe, das Umgebungsmodell mit einer extra Bedingung als gesuchte Funktion zu definieren.

$$f(s,a) = \begin{cases} (s,a) \text{ nicht in Umgebungsmodell} & \text{(s, a) ist gültig} \\ (s,a) \text{ in Umgebungsmodell gültig} & \text{(s, a) ist gültig} \\ (s,a) \text{ in Umgebungsmodell ungültig} & \text{(s, a) ist ungültig} \end{cases} \tag{3.5}$$

Das Umgebungsmodell dient somit als *Lookup-Tabelle* zur Bestimmung der Gültigkeit eines gegebenen (Zustand, Aktion)-Paares. Ist das gegebene (Zustand, Aktion)-Paar noch nicht im Umgebungsmodell enthalten, so wird es vorerst als gültig definiert um dem Reinforcement Learning und dem *on-the-fly*-Verfahren die Möglichkeit zu geben, diese Aktion auszuführen, um sie anschließend dem Umgebungsmodell hinzuzufügen woraufhin sie beim nächsten Auftreten korrekt klassifiziert ist.

3.4.2 Algorithmen

Die für beide Ansätze, *on-the-fly* sowie *precomputed*, benötigten Algorithmen werden nun im Folgenden kurz aufgezeigt und erläutert, doch zunächst sei ein erklärendes Wort zu den verwendeten Variablen verloren. Da das Umgebungsmodell ein Mapping von (Zustand, Aktion)-Paaren auf ein Element in $\{gültig, ungültig\}$ darstellt, lässt sich das Umgebungsmodell als Hash-Map abstrahieren und wird im Folgendem als $environment$ bezeichnet. Außerdem wird zur Bestimmung der Gültigkeit der jeweils derzeitige Zustand (als $state$ bezeichnet), der Folgezustand ($state'$) sowie die zu dieser Transition gehörenden Aktion (im Folgenden $action$ genannt) benötigt. Dies sind somit alle Variablen die benötigt werden, um das Umgebungsmodell um das Paar (state, a) zu erweitern und die Gültigkeit des Paares zu definieren. Als Start sei das *„on-the-fly"*-Verfahren beleuchtet.
Der dazugehörige Algorithmus ist sehr einfach und lässt sich im Grunde als einfache Methode implementieren, welche vom System nach jedem Schritt des Reinforcement Learners aufgerufen wird. Die Idee ist wie folgt:

Listing 3.3: On-the-fly Algorithmus

```
1  Initialisiere HashMap<(Zustand, Aktion)-Paar, {gültig, ungültig}> environment
2  So lange das Reinforcement Learning läuft (while(!rl.finished()) {
3     Erwarte Aufruf von Reinforcement Learner mit Tripel (state, action, state')
4     if(environment enthält das Paar (state, action) noch nicht) {
5        if(state == state')
6           environment.put((state, action), ungültig)
7        else
8           environment.put((state, action), gültig)
9     }
10 }
```

Das Verfahren läuft somit also im Hintergrund des Reinforcement Learnings und erwartet Aufrufe, welche die benötigten Informationen als Übergabeparameter enthalten und erweitert somit iterativ das Umgebungsmodell um höchstens ein (Zustand, Aktion)-Paar.

Wenden wir uns nun dem *precomputed*-Verfahren zu. Da sich der Supervised Agent eigenständig im Labyrinth bewegt, werden zwei zusätzliche Variablen benötigt. Die erste zusätzliche Variable stellt die *worklist* dar. Sie enthält alle Zustände, welche es noch zu überprüfen gilt und enthält zu Beginn nur den Startzustand. Sobald sie leer ist, wurden alle möglichen Zustände im Labyrinth entdeckt und überprüft. *Hinweis:* Der Aufruf *worklist.poll()* gibt das erste Element der Liste zurück und entfernt es anschließend aus der Liste, wodurch die *worklist* kleiner wird. Die zweite benötigte Variable ist nun eine Liste aller Zustände, welche bereits besucht worden sind: *visited*. Mit Hilfe dieser Liste von bereits überprüften Zuständen werden Zyklen vermieden, damit das Verfahren gegen ein vollständiges Umgebungsmodell konvergieren kann. Der Algorithmus ist folgendermaßen:

Listing 3.4: Precomputed Algorithmus

```
1  Initialisiere HashMap<(Zustand, Aktion)−Paar, {gültig, ungültig}> environment
2  Initialisiere List<Zustand> worklist, visited
3  Definiere Zustand start = Startzustand, ziel = Zielzustand, state, state'
4  Füge start der worklist hinzu (worklist.add(start))
5  So lange die worklist nicht leer ist (while(!worklist.isEmpty())) {
6     state = worklist.poll()
7     Bewege Supervised Agenten zu Zustand state im Labyrinth
8     if(state == ziel)
9        Überspringe diesen Zustand und fahre mit dem nächsten fort
10
11    for(jede Aktion action ∈ Aktionsmenge A von s) {
12       Führe Aktion action aus und betrachte state' = Folgezustand
13       if(state == state')
14         environment.put((state, action), ungültig)
15       else {
16         environment.put((state, action), gültig)
17         if(state' ∉ visited)
18            Füge state' der worklist hinzu: worklist.add(state')
19       }
20       Führe das Inverse zu Aktion action aus, um in den Ausgangszustand zu gelangen
21    }
22 }
```

Durch das Verfahren bewegt sich der Supervised Agent also von Zustand zu Zustand und führt in jedem alle möglichen Aktionen aus, um die Transitionsmenge der gültigen sowie ungültigen (Zustand, Aktion)-Paare festzustellen. Einzig und allein im Zielzustand führt das Verfahren keine Aktionen aus, da dieser laut den Regeln keine ausgehenden Transitionen besitzt und somit eine natürliche Grenze des Labyrinths darstellt.

3.4.3 Der Nutzen

Nachdem das Umgebungsmodell aufgebaut wurde (*oder iterativ aufgebaut wird*), fallen für das Reinforcement Learning alle ungültigen (Zustand, Aktion)-Paare weg, wodurch sich in der Theorie eine schnellere Konvergenz gegen die optimale Lösung ergeben sollte, da zum Teil erheblich weniger (Zustand, Aktion)-Paare überhaupt zur Lösung in Frage kommen. Folgende Abbildung 3.9 verdeutlicht dies. In diesem einfachem Beispiel wurden insgesamt **225** mögliche (Zustand, Aktion)-Paare aus der zu untersuchenden Menge entfernt, was eine Reduktion der Gesamtmenge von insgesamt 50.33% entspricht.

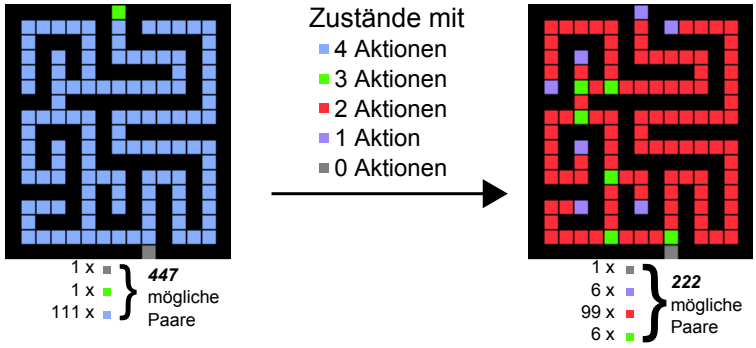

Abbildung 3.9: Auswirkung des Umgebungsmodells auf die mögliche Menge der (Zustand, Aktion)-Paare.

Ein weiterer Vorteil des Umgebungsmodells besteht darin, das im Falle des *precomputed*-Verfahrens bereits zu Beginn des Reinforcement Learnings ein *vollständiges* Abbild des Labyrinths vorhanden ist, welches in das Unsupervised Learning einfließen kann. Durch dieses Abbild ist es dem Unsupervised Learning noch vor dem Start des Reinforcement Learnings möglich zu clustern, wodurch die *Explorationsphase* des Unsupervised Learnings entfällt, da das Umgebungsmodell bereits *alle* Zustände und deren Transitionsmöglichkeiten, die zur Lösung des Problems benötigt werden, enthält.
Daraus resultiert ein in zweierlei Hinsicht reduziertes Problem für das Reinforcement Learning: zum einen ist die Menge der möglichen (Zustand, Aktion)-Paare in den meisten fällen wesentlich reduziert worden, zum anderen ist auch die Zustandsmenge selbst auf das benötigte Minimum beschränkt worden.
Anhand dieser Möglichkeit des *precomputed*-Verfahrens bildet sich folgende Architektur in Abbildung 3.10.

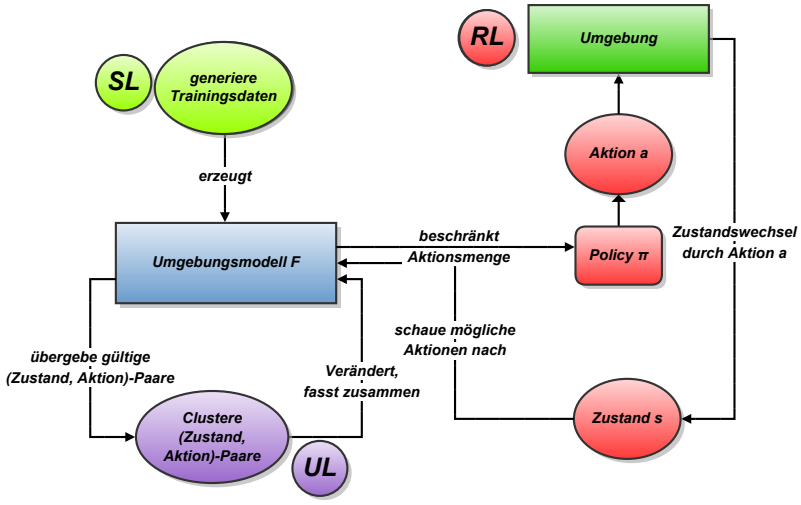

Abbildung 3.10: Verbindung von Supervised, Unsupervised und Reinforcement Learning.

3.5 Verkapseln erlernter Vorgehensweisen

Nachdem nun das Problemszenario mittels Unsupervised und Supervised Learning in seiner Problemdimension reduziert wurde und somit potenziell weniger Zustände als auch weniger (Zustand, Aktion)-Paare zur Lösung betrachtet werden müssen, bleibt die Frage offen, ob dies alle intuitiv möglichen Reduktionen sind. Um diese Frage zu untersuchen sei nochmals Abbildung 3.9 bedacht. Abstrahiert man die Idee eines Labyrinths, so ist es nichts weiter als eine Menge von Gängen welche in verschiedene Richtungen führen und Beschränkungen welche die Wege voneinander trennen. Außerdem gibt es in aller Regel eine Startposition sowie eine oder mehrere Zielposition. Mit diesem Hintergedanken ist der Gedanke nicht all zu abwegig, eine *besondere* Art von Labyrinth zu erkennen: den Straßenverkehr. Betrachten wir einen kleinen Ausschnitt aus Lampertheim in Hessen mittels Google Maps mit einer Start- und Zielposition.

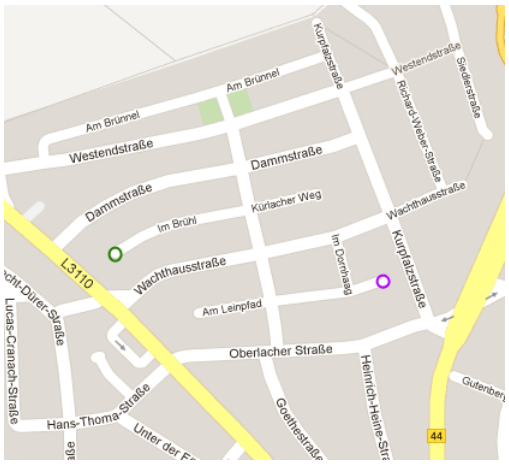

Abbildung 3.11: Ausschnitt einer Google-Maps Karte. Start in grün, Ziel in lila.

Angenommen, man versucht von der gegebenen Startposition zur Zielposition zu gelangen, so ist uns intuitiv klar: wir können nur der Straße folgen und haben keine Entscheidungsgewalt darüber, da wir ansonsten gegen Häuser fahren müssten oder über einen Weg der keine Straße ist (zumindest in aller Regel). Dies ist intuitiv klar und solche Straßen kennt jeder als Sackgassen. Weiterhin angenommen, man kenne die Straßen nicht und fährt drauf los und fährt geradewegs in die Straße Kürlacher Weg (eine weitere Sackgasse). Da es hier auch nicht weiter geht ist ebenfalls sofort bewusst: hier findet ebenfalls keine Entscheidung statt, es bleibt nur der Rückweg. Außerdem wissen wir nun, das uns diese Straßen nichts bringen und versuchen sie zu vermeiden, in dem wir am letzten Punkt, an dem wir eine Entscheidung treffen konnten (z.B. an einer Kreuzung), einen anderen Weg einschlagen. Somit verbleiben in diesem Beispiel nur noch die beiden Möglichkeiten an der Kreuzung Richtung Westendstraße oder Oberlacher Straße zu fahren. Außerdem wird nach mehrmaligen Fahren klar, dass wenn man in Richtung Westendstraße fährt, man einen längeren Weg zur Zielposition eingeht und somit die Richtung Oberlacher Straße als einzige Verbindung in Frage kommt, am schnellsten von der Startposition zur Zielposition zu gelangen. Aus diesem kleinen Exkurs in den Straßenverkehr lassen sich also zweierlei Schlüsse ziehen: Aus Sackgassen führt nur ein Weg (nämlich der Eingang) und es lohnt sich nicht, sie erneut zu besuchen solange sie nicht zur Zielposition führen und außerdem kommen ab einer Anzahl von Versuchen nur noch gewisse Richtungen in Frage, welche schnellstmöglich zum Ziel führen und die anderen werden somit nach und nach vernachlässigbar.
Was so für uns Menschen so intuitiv erscheint ist hingegen für das maschinelle Lernen nicht all zu einfach. Im Falle des Reinforcement Learnings werden mehrere Iterationen benötigt, diese Sackgassen als solche zu erkennen und versucht sie anschließend zu vermeiden, wenn sie nicht zum Lösungsweg beitragen. Dies beinhaltet allerdings das betrachten des Zustands, berechnen des nächsten Schritts anhand der Vorgehensweise, berechnen möglicher Zufallsschritte im Falle des ϵ-Greedy Verfahrens und anschließendes berechnen des neuen Q-Wertes. Diese Iterationen sind allerdings in vielen wie zuvor gezeigten Fällen unnötig und verursachen Rechenaufwand, der unter Umständen an anderen Stellen benötigt wird, oder (wie in diesem Fall) eine schnellere Konvergenz gegen die optimale Lösung erzielen könnte. Eine Möglichkeit diesen unnötigen Rechenaufwand zu vermeiden wurde bereits in Abschnitt 2.4 vorgestellt: das *Verkapseln erlernten Wissens*.

3.5.1 Umsetzen der Idee

Nach der obigen Analyse einer Stadtkarte wurden zwei weitere Möglichkeiten entdeckt, das Problem weiter zu reduzieren. Es stellt sich nur die Frage: wie lässt sich dies erreichen? Die Grundlegende Idee zur Umsetzung wurde bereits in Abschnitt 2.4 beschrieben: wenn der Unterschied zwischen zwei Iterationen bezüglich eines Zustands geringer als ein gegebener Schwellwert x ist, wird dieser Zustand auch mit den folgenden Iterationen kein neues Wissen dazulernen und der Lernalgorithmus degeneriert von einer Exploration zu einer Exploitation. Ab diesem Punkt lässt sich dieser Zustand mit samt seines bisher gelernten Wissens aus dem Lernalgorithmus nehmen und einzeln betrachten (*„Verkapselt"* als Zustand mit zugehöriger *Standardaktion*). Im Laufe der Iterationen löst sich somit der oben beschriebene Mehraufwand für unnötige Zustände nach einer Zeit von alleine auf, da für jeden Zustand nach und nach Standardaktionen definiert werden und nicht mehr vom Lernalgorithmus untersucht werden müssen. Somit sind direkt auf einen Streich (*genug Iterationen vorausgesetzt*) zwei Punkte abgehakt: zum einen werden Richtungen von Iteration zu Iteration zunehmend klarer und benötigen keinen Rechenaufwand mehr und zum anderen resultiert aus dieser Eigenschaft das Auffinden von Sackgassen. Da am Ende einer Sackgasse nur noch eine Aktion (die Rückrichtung) möglich ist, wird diese im Verlauf der Lernphase einen erhöhten Q-Wert erhalten und schneller als Standardaktion die Rückrichtung definieren.
Was nun noch allerdings fehlt, ist der Schluss aus einer solchen Standardaktion. Sei Abbildung 3.12 betrachtet, auf dem ein einfacher Zustandsübergangsgraph mit den vier Zuständen A, B, C und D und ihren jeweiligen Transitionen zu sehen ist, wobei Zustand A der Start und Zustand C das Ziel ist. Die Analogie einer Sackgasse sieht man am Zustand A, welcher lediglich die Transition nach Zustand B besitzt, folglich können wir diese als Standardtransition definieren. Doch was ist nun mit der Transition $B \rightarrow A$? Diese Transition ist nach der Analyse der Straßenkarte aus der Einführung dieses Kapitels zu verwerfen, da sie zu einer Schleife führen wurde und intuitiv betrachtet einen auch nicht weiter bringen kann, schließlich würde man nur zwischen zwei Zuständen hin und her springen. Daraus folgt wiederum, dass Zustand B nur noch eine Transition aufweist – die nach Zustand D. Hier spielt sich das gleiche ab: definiere für Zustand B die Standardtransition $B \rightarrow D$ und betrachte nicht mehr die Transition $D \rightarrow B$, da diese keinen Nutzen mehr bringt. Als Ergebnis folgt die rechte Seite der Abbildung: ein gerichteter, azyklischer Graph mit nur einer Richtung, welche direkt vom Startzustand A zum Zielzustand C führt, ohne auch nur eine Iteration der Lernphase zu benötigen.

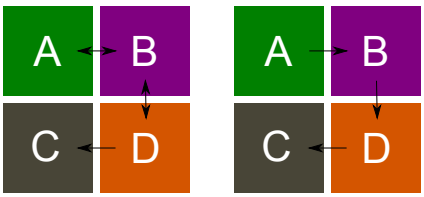

Abbildung 3.12: Graph vor und nach Auswirkung der Sackgasse.

Die getroffenen Beobachtungen lassen sich nun in vier Kriterien zur Definition von Standardaktionen zusammenfassen:

1. Besitzt die Aktionsmenge A_s des Zustands s nur eine *gültige* Aktion, so definiere diese Aktion $a \in A_s$ als Standardaktion.

2. Haben genug Iterationen auf einen Zustand s stattgefunden, so ist dieser Zustand bereit eine Standardaktion zu definieren. Ist die Anzahl der Iterationen zu gering, so ist keine Aussage zu treffen, da es potenziell noch nicht besuchte Wege gibt.

3. Ist ein Zustand s bereit für eine Standardaktion und ist eine Aktion a *deutlich* besser als alle anderen, so definiere sie als Standardaktion.

4. Entferne aus jedem Zielzustand einer Standardaktion das Inverse zu dieser Aktion, sodass das Ergebnis einen gerichteten Teilgraphen ergibt.

Nun, nachdem die einzelnen Kriterien definiert wurden, welche über die Definition einer Standardaktion entscheiden, folgt die Umsetzung mit Hilfe des Reinforcement, Unsupervised und Supervised Agenten.
Es fällt auf, dass das erste Kriterium eine Aussage über die Anzahl der gültigen Aktionen der jeweiligen Aktionsmenge benötigt und dementsprechend die Standardaktion definiert. Diese Informationen stehen allerdings nur beim Supervised Learning zur Verfügung weshalb man die Umsetzung in zwei Kategorien unterteilen kann: zum einen Verkapselung ohne Umgebungsmodell, zum anderen Verkapselung mit Hilfe eines Umgebungsmodells.

3.5.2 Verkapselung ohne Umgebungsmodell

Wenn die Verkapselung ohne die Erweiterung eines wie in Abschnitt 3.4 beschriebenen Umgebungsmodells bearbeitet wird, stehen wie zuvor schon beschrieben die Informationen über die Mächtigkeit der einzelnen Aktionsmengen A_s zum jeweiligen Zustand s nicht zur Verfügung. Somit beruht das Verfahren zur Verkapselung lediglich auf den Iterationen, welche im Verlauf der Lernphase des Reinforcement Agenten auf den einzelnen Zuständen stattfinden. Daraus folgt, dass das erste Kriterium für diese Herangehensweise nicht von Bedeutung ist und im Folgenden wegfällt. Aus diesem Grund seien nun die Kriterien 2 sowie 3 näher betrachtet um ihre Intuition zu betrachten sowie sie zum einen auf theoretische Korrektheit sowie Nutzen zu untersuchen. Sei im Folgenden mit dem 2. Kriterium begonnen.
Das 2. Kriterium definiert die Situation, in welchem sich die Lernphase auf den zu Untersuchenden Zustand s befinden muss, um das definieren einer Standardaktion zu ermöglichen. In ihm wird definiert, dass eine Mindestanzahl von Iterationen auf einen Zustand benötigt werden, um überhaupt eine Grundlage zur Bestimmung der Standardaktion zu erhalten. Die Intuition dahinter ist zunächst das Labyrinth mehrere male zu lösen um etwaige Lösungswege zu finden, um anschließend ein Urteil bezüglich einer Standardaktion fällen zu können. Verwirft man zu früh andere Möglichkeiten, so besteht die Gefahr potenziell den kürzesten Weg vom Start- zum Zielzustand zu verwerfen. Dieses Kriterium wird somit benötigt, um einen Problemfall abzudecken, welcher anhand folgender Abbildung verdeutlicht werden soll.

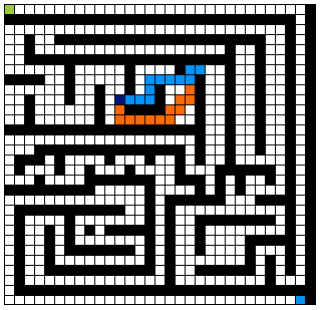

Abbildung 3.13: Entscheidungsproblem: Blau oder Orange?

Obige Abbildung stellt die Situation dar, welche bei der Iteration auf dem dunkel-blauen Zustand stattfindet. Angenommen der Reinforcement Agent hat in den Vergangenen vier Iterationen zuerst den orangen Weg dreimal eingeschlagen und den blauen Weg anschließend nur einmal. Nun stellt sich die Frage, ob der orange Weg besser als der blaue ist, denn schließlich wurde er dreimal so oft verwendet wie der blaue weg. *Nein* ist in diesem Fall die klare Antwort, da der orange Weg eine Länge von 15 Schritten, der blaue hingegen nur eine Länge von 13 Schritten aufweist. Angenommen man lässt die Lernphase nun weitere 100 Iterationen machen, so würde sich ein klareres Bild ergeben, welcher der beiden Wege vom Reinforcement Learning bevorzugt wird. In Folge dessen ist eine Mindestanzahl von Iterationen nötig und wird im Folgenden als *Schwellwert* x bezeichnet, außerdem wird im gleichen Atemzug $I(s)$ als die Anzahl der Iterationen auf Zustand s definiert.
Sei nun der Fall betrachtet, dass mindestens x Iterationen auf Zustand s stattgefunden haben, sodass das 4. Kriterium greift. Auch hier wird ein Schwellwert y benötigt, um ein Urteil bilden zu können, ob eine Aktion a im Zustand s tatsächlich als Standardaktion in Frage kommt. Als Standardaktion ist diejenige Aktion zu definieren, welche im Durchschnitt am häufigsten verwendet wird und der Schwellwert y entscheidet im Folgenden über die Höhe dieses Durchschnitts gesuchten Durchschnitts. Sei als Beispiel ein wie in dieser Arbeit verwendetes ϵ-Greedy Verfahren zur Bestimmung der nächsten Aktion gewählt. Ist ϵ hoch genug, kann es durchaus vorkommen, dass obwohl genug Iterationen auf einen Zustand stattgefunden haben, keine Aussage über eine Standardaktion getroffen werden kann. Ist ϵ entsprechend hoch lässt sich eine Situation finden, in der auf einem Zustand s die Aktion a_1 51 mal gewählt wurde und Aktion a_2 hingegen 49 mal. Im Durchschnitt über 100 Iterationen gesehen scheint a_1 besser als a_2 zu sein, doch angenommen a_2 führt zu einer kürzeren Lösung als a_1, so kann es gefährlich sein a_1 als Standardaktion zu definieren, da ab diesem Punkt der Lernalgorithmus keinen Einfluss mehr auf diesen Zustand und seiner Standardaktion hätte. Folglich ist es sicherer, den zuvor beschriebenen Schwellwert y derart zu verwenden, dass Aktion a_1 genau dann als Standardaktion definiert wird, wenn $\frac{A(s,a_1)}{A(s,a_2)} > y$ gilt, wobei $A(s,a)$ angibt, wie oft Aktion a in Zustand s im Verlauf der Lernphase gewählt wurde.
Mittels dieser beiden Schwellwerte x und y lässt sich also eine Aktion als Standardaktion definieren, wenn kein Umgebungsmodell die Suche nach Standardaktionen unterstützt.

3.5.3 Verkapselung mit Umgebungsmodell

Sei nun das Verfahren der Verkapselung mit Hilfe des Umgebungsmodells betrachtet, welches durch das in Abschnitt 3.4 beschriebene Verfahren entsteht. Mit Hilfe des Umgebungsmodells, welches lediglich die gültigen Transitionen der einzelnen Zustände betrachtet, ist eine Aussage über die Anzahl der gültigen Aktionen des jeweiligen Zustands s vorhanden und das erste Kriterium hat die Möglichkeit Sackgassen bereits ohne eine Iteration auf einen solchen Sackgassen-Zustand zu erkennen. Dies hat zur Folge, dass das 4. Kriterium (*Entfernen der Zyklen*) besonders stark zur Geltung kommt und das Labyrinth, welches typischerweise viele enge Gänge und Sackgassen aufweist, erheblich vereinfacht werden kann und die Konvergenz somit beschleunigt wird. Dies geschieht in den meisten Fällen ohne eine einzige Iteration des Lernalgorithmus auf den jeweiligen Zuständen, welche somit eine Standardaktion definieren können und die Aktionsmengen ihrer Zielzustände weiter einschränken, wodurch weitere Entscheidungsmöglichkeiten aus dem Suchraum entfernt werden. Somit reduziert jede gefundene Sackgasse die Problemdimension signifikant und es müssen nur noch Zustände mittels des Lernalgorithmus untersucht werden, welche die Möglichkeit einer Entscheidung zwischen mindestens zwei Richtungen aufweisen. Um die Effizienz dieser beiden Kriterien zu betrachten, sei folgende Abbildung betrachtet. Sie verdeutlicht die einfache Anwendung des 1. sowie 4. Kriteriums in abwechselnder Reihenfolge und die daraus resultierende Bedeutung für das zu untersuchende Labyrinth.

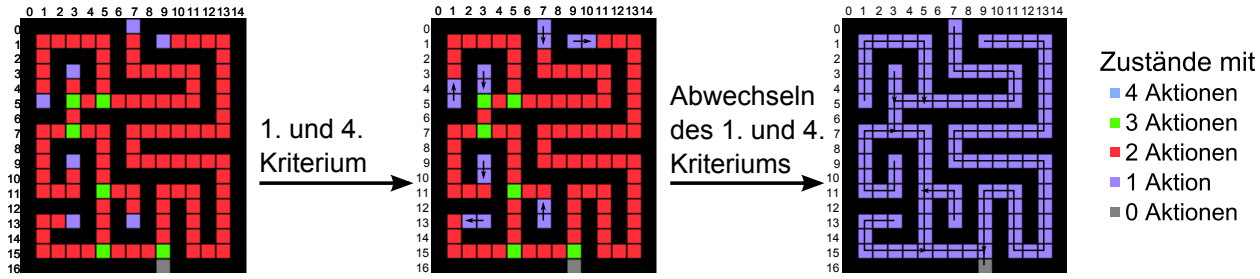

Abbildung 3.14: Anwenden des ersten und vierten Kriteriums.

Im ersten Teil der Abbildung ist die Ausgangslage des Umgebungsmodells gezeigt mit den Sackgassen (*Zustände mit nur einer Richtung / Aktion*) golden hervorgehoben. Die Startposition des Agenten liegt bei $(7,0)$.
Betrachten wir nun im folgenden die Vorgehensweise exemplarisch anhand dieser Abbildung. Da der Anfangszustand nur eine Aktion aufweist, greift Kriterium 1: der Lernalgorithmus braucht keine Iteration auf diesem Zustand auszuführen und als Standardaktion wird $\downarrow$ festgelegt. Nun greift Kriterium 4 und in Zustand $(7,1)$ wird die Aktion $\uparrow$ gelöscht. Das Resultat ist, dass Zustand $(7,1)$ ebenfalls nur noch eine Richtung aufweist und für diesen Zustand entsprechend wiederum Kriterium 1 greift. Nun wechseln sich die beiden Kriterien zunehmend ab, sodass jeder Zustand im Pfad von Zustand $(7,0)$ bis $(6,5)$ eine Standardaktion aufweist und bisher keine Iteration des Agenten von Nöten war. Aus Kriterium 4 folgt für Zustand $(5,5)$, dass die Aktion $\rightarrow$ entfernt wurde und dieser Zustand nun eine 2-elementige Aktionsmenge $A_{5,5} = \{\uparrow, \leftarrow\}$ aufweist. Da dieser Zustand 2 gültige Aktionen hat, greift dieses mal Kriterium 1 nicht und es ist eine Iteration des Lernalgorithmus von Nöten. Im Verlauf der Iterationen wird der Agent irgendwann die Aktion $\uparrow$ ausführen und dem Gang Richtung Zustand $(1,5)$ folgen. Dies hat zur Folge, dass jeder Zustand im Gang von Zustand $(1,5)$ bis hin zu Zustand $(5,4)$ durch die abwechselnde Verwendung der Kriterien 1 und 4 eine Standardaktion definiert bekommt und die Aktionsmenge $A_{5,5}$ somit weiter auf $\{\leftarrow\}$ eingeschränkt wird und folglich wieder Kriterium 1 greift. Führt man dieses Schema fort, sieht man das Resultat im 3. Teil der Abbildung. Dieser Teil stellt das Labyrinth dar, nachdem alle Zustände abwechselnd mit den Kriterien 3 sowie 4 bearbeitet wurden und an den zuvor orangen Zuständen Richtungsentscheidungen von Nöten waren. Das Resultat ist in diesem einfachem Beispiel deutlich bemerkbar, da dank der Sackgassen (*sobald jeweils einmal erreicht*) insgesamt 97 weniger Zustände vom Lernalgorithmus zu untersuchen sind.
Somit bildet die Verkapselung mit Umgebungsmodell ein bedeutendes Hilfsmittel in Labyrinthen, welche Sackgassen verbunden mit engen Gassen aufweisen. Sind diese allerdings nicht vorhanden, entspricht die Performanz zu Beginn der Lernphase dem des Verfahrens ohne Umgebungsmodell. Im späteren Verlauf hingegen, nachdem Standardaktionen definiert wurden und anhand des vierten Kriteriums die Aktionsmengen ihrer jeweiligen Zielzustände eingeschränkt hat, beginnt dieses Verfahren potenziell mehr Rechenaufwand als die Variante ohne Umgebungsmodell einzusparen. Der Grund hierfür liegt in der Tatsache begründet, dass durch die zusätzliche Einschränkung der Aktionsmengen Zustände entstehen könnten, welche nur noch eine gültige Aktion aufweisen und somit keine weiteren Iterationen mehr benötigen. Dementsprechend ist dieses Verfahren, wenn möglich, dem zuvor in Abschnitt 3.5.2 beschriebenen vorzuziehen.

3.5.4 Implementierung

Nachdem nun beide Varianten der Verkapselung besprochen wurden, ist es an der Zeit die Implementierung beider zu betrachten. Im Grunde entspricht die Implementierung beider Verfahren einer Erweiterung des Reinforcement Learners, welche zwei Punkte enthält: zum einen die Abfrage, ob ein Zustand eine Standardaktion definiert hat und zum anderen, falls keine definiert wurde, die Überprüfung der Kriterien 1 bis 4. Das Gerüst der Implementierung stellt somit die Reinforcement Learning Methode selbst dar, mit einer Zusätzlichen *if-Bedingung* vor der eigentlichen Bestimmung der zu wählenden Aktion a in Zustand s, sowie zwei extra Aufrufe der Methode *encapsulateAction(s)*, welche auf dem übergebenen Zustand s die Kriterien 1 bis 4 überprüft und entsprechend der Erfüllung eines Kriteriums eine Standardaktion definiert oder nicht. Das Gerüst ist somit das in Listing 3.5 zu sehende.

Listing 3.5: Gerüst zur Anwendung der Verkapselung

```
1  Reinforcement Agent wählt Zustand s
2  if(s definiert Standardaktion a oder besitzt nur eine gültige Aktion a)
3     s = Folgezustand s' der in Zustand s mittels Aktion a erreicht wird
4     encapsulateAction(s)
5  else {
6     Berechne anhand der Vorgehensweise die nächste Aktion a
7     Führe Aktion a aus, betrachte Folgezustand s'
8     Berechne Q–Werte
9     Setze Folgezustand s' und nächste Aktion a'
10    encapsulateAction(s)
11 }
```

Die Methode *encapsulateAction(s)* ist dabei wie folgt anhand der Kriterien 1 bis 4 auf Seite 32 implementierbar. Zur Erinnerung: $I(s)$ beschreibt die Anzahl der Iterationen auf Zustand s, $A(s, a)$ beschreibt hingegen wie oft Aktion a in Zustand s verwendet wurde und zu guter Letzt stellen x und y die auf Seite 33 definierten Schwellwerte der Kriterien 2 und 3 dar.

Listing 3.6: Implementierung der encapsulateAction-Methode

```
1  encapsulateAction(Zustand s) {
2     if(s besitzt nur eine Aktion a) (Kriterium 1)
3        Definiere a als Standardaktion von Zustand s
4     if(I(s) > x) { (Kriterium 2)
5        Finde die beiden Aktionen a1 und a2 welche am häufigsten in Zustand s
6        gewählt wurden, wobei a1 häufiger als a2 vorkam
7        Berechne diff = A(s, a1) / A(s, a2)
8        if(diff > y) (Kriterium 3)
9           Definiere a1 als Standardaktion von Zustand s
10    }
11    if(s hat Standardaktion a definiert) { (Kriterium 4)
12       s' = Folgezustand s' der in Zustand s mittels Aktion a erreicht wird
13       a' = Inverse zu Aktion a, bspw. zu a = → ist a' = ←
14       Entferne Aktion a' aus der Aktionsmenge von s'
15    }
16 }
```

4 Experimentelles Setup

Nachdem nun das Problemszenario in Abschnitt 1.2 beschrieben wurde und in Kapitel 3 alle verwendeten Algorithmen samt ihrer zu Grunde liegenden Ideen erläutert wurden, beschäftigt sich dieser Abschnitt mit den möglichen Kombinationen dieser Algorithmen, welche im weiteren Rahmen des Experiments verwendet werden.
Hierbei wurde das Problemszenario in 4 allgemeine Arten von Labyrinthe eingegliedert:

1. *klassische* Labyrinthe. Diese Labyrinthe weisen eine Vielzahl an Sackgassen auf und sind von ihrer Art der Gänge sehr eng (1 Zustand weit). Hierbei gibt es oftmals nur einen korrekten Weg zum Ziel.

2. *offene* Labyrinthe. Diese Art der Labyrinthe dient der Betrachtung des *worst-case*-Szenarios, da sich kaum Optimierungsmöglichkeiten anbieten. In ihnen gibt es nur eine niedrige Anzahl an Mauern und der Weg von Start nach Ziel ist relativ frei wählbar.

3. *freie* Labyrinthe. Labyrinthe welche in diese Klasse fallen haben einen breiteren Gang und stellen somit ein Problem bei der trivialen Auffindung von Sackgassen dar. Außerdem gibt es mehrere Wege zum Ziel.

4. *knifflige* Labyrinthe. Diese Labyrinthe stellen eine Vereinigung der drei obigen Labyrinth-Arten dar. In ihnen gibt es mehrere Wege zum Ziel welche nahezu gleich in ihrer Länge sind und außerdem breitere Gänge aufweisen. Auch können freie Areale im Labyrinth enthalten sein.

Anhand dieser vier Arten von Labyrinthen sollten die meisten herkömmlichen Labyrinthe abgedeckt und verschiedene Hürden vorhanden sein, sodass sich die erzielten Resultate der verschiedenen Kombinationen der Lernalgorithmen entsprechend verändern sollte. Den Ausgangspunkt eines jeden Experiments bildet das Reinforcement Learning (ohne Erweiterungen) und wird mit den jeweils beschriebenen Verfahren Schrittweite erweitert. Hierbei wurden auf allen Labyrinthen, mit der jeweils zu untersuchenden Kombination von Lernverfahren, 32 Durchläufe mit je 1.000, oder im Falle des freien 4.000 und des kniffligen Labyrinths 10.000, Episoden ausgeführt und anschließend der Durchschnitt dieser Daten berechnet. Die verwendeten Parameter in allen Berechnungen waren: $\alpha = 0.1$, $\gamma = 0.98$ sowie $\epsilon = 0.1$. Die Parameter des vorgestellten Verfahrens *Verkapseln erlernten Wissens* werden an der nötigen Stelle beschrieben und unterscheiden sich in den verwendeten Labyrinthen.

Folgende Abbildung liefert zu jeder Art ein Labyrinth dieser Klasse und zeigt zeitgleich die für die kommenden Untersuchungen verwendeten Labyrinthe. Diese Labyrinthe werden in den kommenden Abschnitten anhand ihrer Arten benannt, sodass wenn vom *klassischen* Labyrinth die Rede ist, das Labyrinth links in Abbildung 5.1 gemeint ist, beim *kniffligen* Labyrinth hingegen vom rechten Labyrinth die Rede ist.

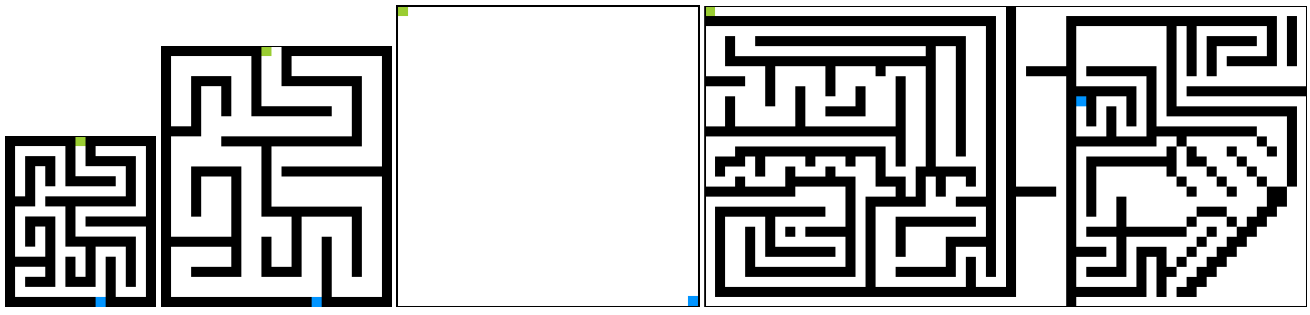

Abbildung 4.1: Veranschaulichung der einzelnen Labyrinth-Arten der Reihe nach von links nach rechts.

4.1 Verwendete Architekturen

Im Rahmen des 3. Kapitels wurden die drei Lernparadigmen *Reinforcement*, *Unsupervised* sowie *Supervised Learning* vorgestellt, doch es bleibt die Frage offen, wie diese Ideen miteinander verknüpft werden können. Dieser Abschnitt widmet sich den in diesem Experiment verwendeten Architekturen und erläutert die Intention und Vorgehensweise. Sei zunächst mit der Basis, dem Reinforcement Learning, begonnen.

Reinforcement Learning

Als Basis des Experiments dient das Reinforcement Learning ohne Erweiterungen, dessen Resultate im weiteren Verlauf des Experiments als Referenzwerte fungieren. Wie zuvor in Abschnitt 2.1 beschrieben beinhaltet dieses Lernparadigma einen *Agenten*, welcher auf Zuständen s mittels Aktionen a der jeweiligen Aktionsmenge A_s agiert und dabei Belohnungen r erhält. Die daraus resultierende Architektur ist im Grunde eine *actor-critic*-Architektur wie sie zuvor bereits in Abbildung 2.5 zu sehen war.

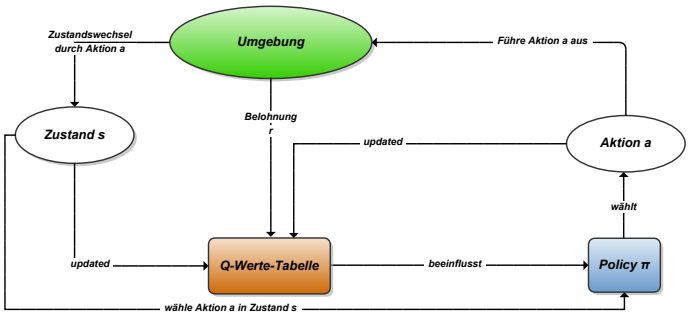

Abbildung 4.2: Verwendete Architektur des Reinforcement Learnings.

Anhand dieser Architektur wählt der Agent somit zu einem gegebenen Zustand s_t gemäß der Vorgehensweise (*Policy*) π basierend auf den gegebenen Q-Werten eine Aktion a_t, führt diese Aktion a_t aus und verändert anschließend mittels dieses Tripels (s_t, a_t, r_t) den dazugehörigen Q-Wert $Q(s_t, a_t)$ gemäß der gegebenen Update-Regel aus Gleichung 3.1.

Reinforcement Learning erweitert mit Unsupervised Learning

Die erste Erweiterungen stellt das Unsupervised Learning dar. Mittels des vorgestellten Verfahrens in Abschnitt 3.3 wird jedes (s, a)-Paar an das Verfahren übergeben und anschließend geclustert sobald die Explorationsphase vorüber ist. Anhand dieses Clusterings wird anschließend die Aktionsmenge des Zustands durch Makro-Aktionen erhöht und das Makro-Q-Learning agiert analog zu der zuvor vorgestellten Architektur des reinen Reinforcement Learnings. In dieser Abbildung und den folgenden Abbildungen ist die Beeinflussung der Vorgehensweise durch die Q-Werte-Tabelle (*welche nun sowohl (Zustand, Aktion)-Paaren als auch (Zustand, Makro-Aktion)-Paaren Werte zuweist*) weggelassen worden, um die Übersicht zu erhöhen.

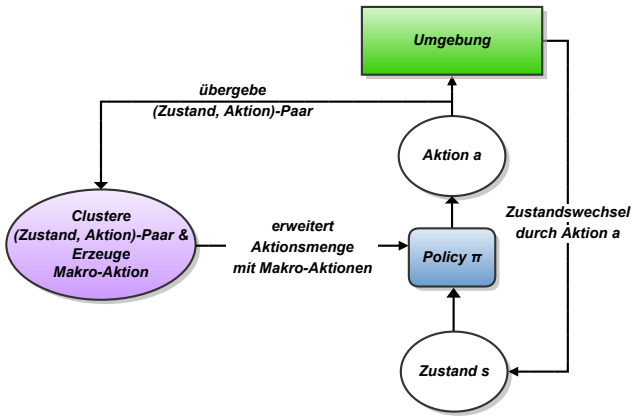

Abbildung 4.3: Verwendete Architektur der Erweiterung mittels Unsupervised Learnings.

Reinforcement Learning erweitert mit Supervised Learning

Die zweite Erweiterung stellt zunächst das Supervised Learning dar. In dieser Kombination erstellt das Supervised Learning zunächst ein Umgebungsmodell (*siehe Abschnitt 3.4*), welches als Ergebnis alle gültigen als auch ungültigen (Zustand, Aktion)-Paare beinhaltet und somit ein Mapping $f(s, a) \rightarrow \{\text{gültig}, \text{ungültig}\}$ ermöglicht. Mittels dieser Mapping-Funktion wird die Aktionsmenge des Reinforcement Agenten eingeschränkt, sodass nur noch gültige Aktionen ausgeführt werden und somit Mauerzustände sowie die damit verbundenen Bestrafungen umgangen werden.

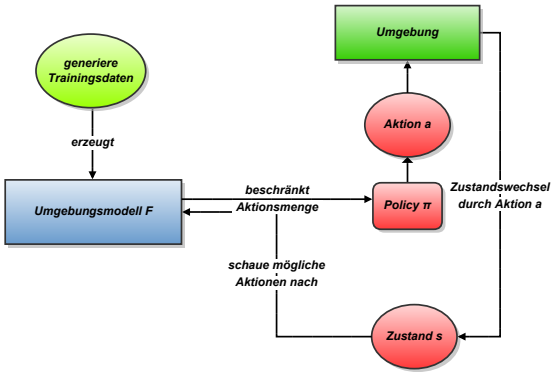

Abbildung 4.4: Verwendete Architektur der Erweiterung mittels Supervised Learnings.

Reinforcement Learning erweitert mit Supervised sowie Unsupervised Learning

Die dritte Architektur stellt eine Kombination aller drei Lernparadigmen dar. In ihr agiert der Reinforcement Agent ebenfalls auf der realen Umgebung, bekommt allerdings zum einen die Aktionsmenge wie in der zuvor vorgestellten Architektur auf gültige Aktionen beschränkt und zum anderen wird die Aktionsmenge ebenfalls um Makro-Aktionen erweitert, falls die Zustände ein Clustering des vorgestellten Unsupervised Verfahrens ermöglichen.

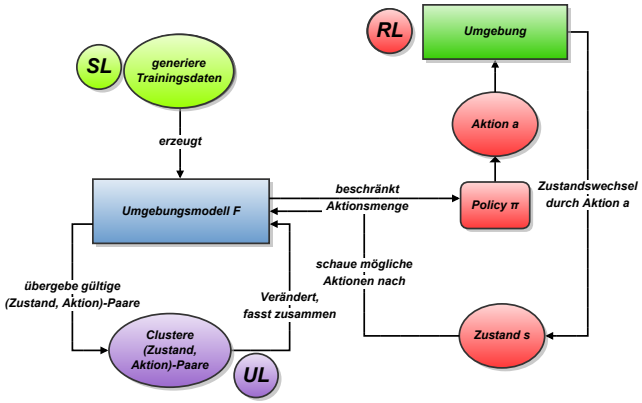

Abbildung 4.5: Verwendete Architektur der Erweiterung mittels Supervised und Unsupervised Learning.

Zusätzliche Erweiterung mittels Verkapselung

Die letzte verwendete Architektur ist eine zusätzliche Erweiterung der Kombination aller drei Lernparadigmen mittels des aus Abschnitt 2.4 kennengelernten *Verkapseln erlernten Wissens*. Die Erweiterung beinhaltet 2 Elemente: Sind zum einen die Kriterien erfüllt, so wird intern im Rahmen des Makro-Q-Learnings eine Standardaktion definiert. Beinhaltet dieser Zustand ebenfalls eine Makro-Aktion, welche die gleiche Richtung wie die Standardaktion einschlägt, so wird diese Makro-Aktion als Standard-Makro-Aktion definiert. Ist hingegen zum anderen bereits eine Standardaktion definiert, so wird diese ohne Auswahl mittels Vorgehensweise π verwendet und weiterer Rechenaufwand ist auf diesem Zustand nicht weiter nötig. Definiert der Zustand neben einer Standardaktion eine Standard-Makro-Aktion, so wird diese bevorzugt verwendet, da sowohl die Standardaktion als auch die Standard-Makro-Aktion die selbe Richtung einschlagen und somit, im weiteren Lernverlauf gesehen, auf dem gleichen Zustand s_{t+n} enden.

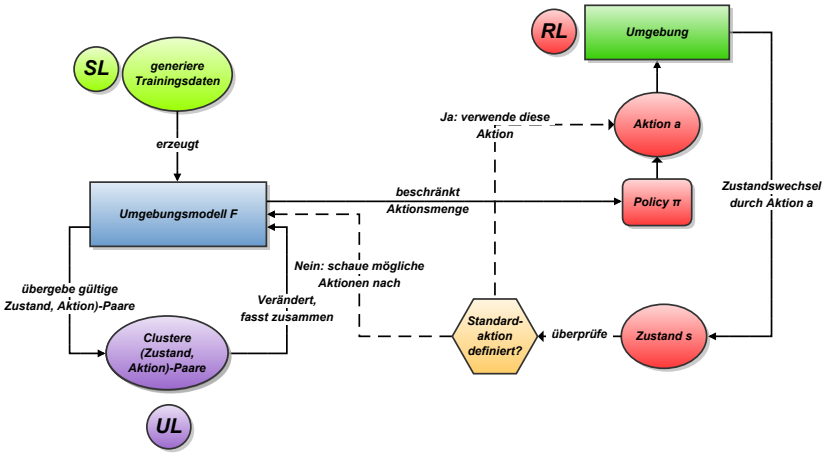

Abbildung 4.6: Erweiterung der Architektur aus Abbildung 4.5 mittels Verkapselung aus Abschnitt 2.4.

5 Ergebnisse

Nun, nachdem alle hier verwendeten Verfahren und Architekturen vorgestellt wurde, werden die erzielten Ergebnisse der einzelnen Verfahren betrachtet. Hierbei wird jede in Abschnitt 4 vorgestellte Labyrinth-Art einzeln für sich betrachtet und anhand dieser jede zuvor vorgestellte Architektur evaluiert. Alle hier verwendeten Graphen, welche die Summe der durchschnittlich erhaltenen Belohnungen pro Episode zeigen, besitzen als Startwert einen *ungelernten* Datensatz und erst ab der 1. Episode sind die tatsächlich erzielten Resultate des jeweiligen Verfahrens zu sehen. Sei zunächst mit dem *klassischen* Labyrinth begonnen.

5.1 klassische Labyrinth

Die Kategorie des klassischen Labyrinths beinhaltet, wie zuvor in Abschnitt 4 beschrieben, viele enge (*1-Zustand breite*) Gänge, Sackgassen sowie in aller Regel nur einen gültigen Weg vom Start- zum Zielzustand.

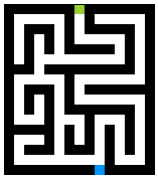

Abbildung 5.1: Das untersuchte Labyrinth aus der Familie der *klassischen* Labyrinthe.

Erwartungshaltung

Aus diesem Umstand heraus folgt somit, dass das Clustering des vorgestellten Unsupervised Verfahrens aus Abschnitt 3.3 auf Seite 22 den Zustandsübergangsgraphen verkleinern und anschließend Makro-Aktionen definieren kann. Somit sollte Clustering bereits einen bemerkbaren Unterschied bezüglich der Konvergenz erzielen, da der Agent keine Fehltentscheidungen treffen kann, während er bereits eine Makro-Aktion ausführt und außerdem die $Q(s, a)$-Werte der Zwischenzustände ebenfalls aktualisiert. Des Weiteren ist zu erwarten, dass der Agent viele Fehlentscheidungen trifft, da durch die Charakteristik der engen Gänge im Durchschnitt betrachtet jeder Zustand zwei Aktionen enthält, welche den Agenten gegen eine Mauer laufen lassen und somit eine Bestrafung mit sich führen. Aus diesem Fakt heraus sollte auch das Umgebungsmodell des Supervised Learnings eine erhebliche Verbesserung mit sich bringen, da nur noch die gültigen Zustandsübergänge betrachtet werden. Zusätzlich sollte das Verfahren des Verkapselns eine nahezu sofortige Konvergenz des Agenten mit sich führen, da die Sackgassen einen erheblichen Einfluss auf die weiteren Lernphasen des Agenten ausüben.

Die hier erzielten Referenzwerte sind in den folgenden Diagrammen dargestellt. Dabei zeigt der linke Graph in Abbildung 5.2 die durchschnittlich erhaltene Summe von Belohnungen pro Episode, hingegen der rechte Graph die durchschnittliche Anzahl der benötigten Schritte, bis der Agent das Ziel erreicht hat. Die dritte und letzte Abbildung hingegen zeigt an, welche Zustände im Rahmen von 10 Lernepisoden wie oft besucht wurden. Die Farbleiste unter dieser Abbildung zeigt hierbei farblich an, welche Anzahl von Zugriffen auf welche Farbnuance abgebildet wird.

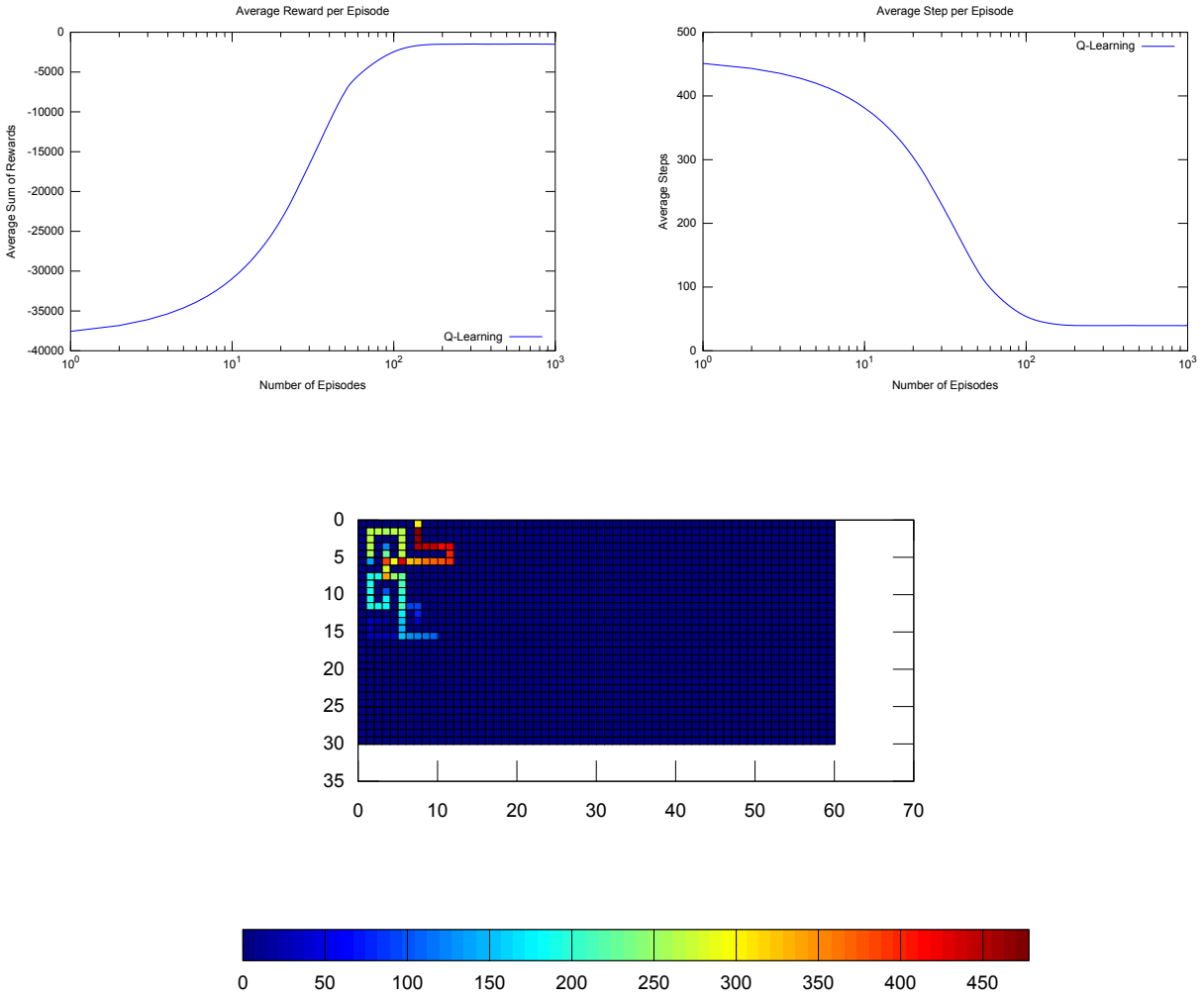

Abbildung 5.2: Daten des Reinforcement Learnings auf das *klassische* Labyrinth.

Hierbei ist festzuhalten, dass das Reinforcement Learning im *klassischen* Labyrinth ungefähr ab der 110 Episode den optimalen Weg vom Start- zum Zielzustand gefunden hat und hierbei eine ausgewogene Mischung aus Exploration sowie Exploitation stattfindet, da alle zur Lösung relevanten Zustände bereits mindestens einmal besucht worden sind und sich die markanten Zustände des optimalen Weges bereits hervorheben. Lediglich der gesamte Pfad rechts vom Ziel wurde bereits verworfen, da dieser Weg nicht kürzer sein kann, als der bereits gefundene.
Als erste Erweiterung sei nun das *Unsupervised Learning* betrachtet.

Den Erwartungen nach sollte der Agent durch den Zusatz von Makro-Aktionen im Durchschnitt betrachtet eine höhere Summe von Belohnungen erhalten und somit als direktes Resultat weniger Schritte benötigen, um das Ziel zu erreichen. Sei Abbildung 5.4 betrachtet, welche die erzielten Resultate der Kombination Q-Learning mit Makro-Aktionen denen des reinen Q-Learnings gegenüberstellt. Hierbei sei angemerkt, dass alle folgenden Graphen mit *Clustering* die Erweiterung des Unsupervised Learnings und damit die Verwendung von Makro-Aktionen beschreibt.

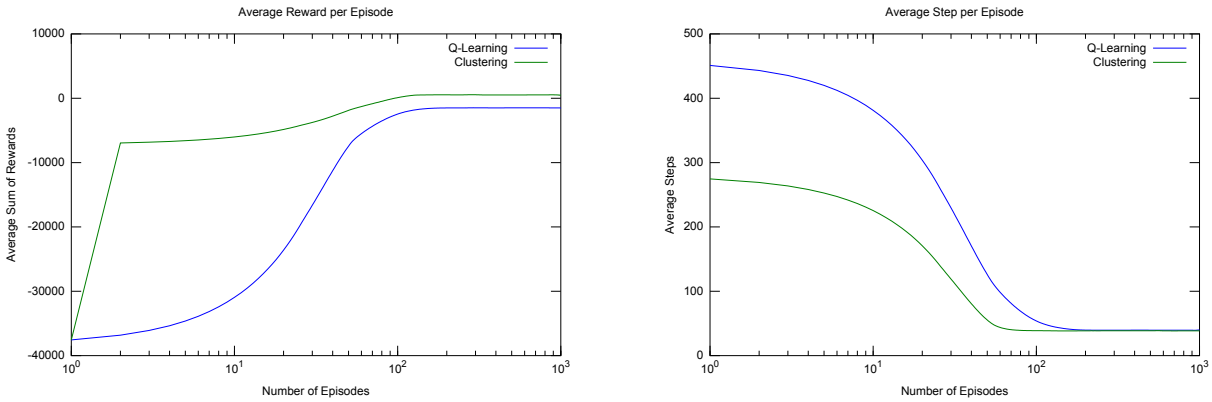

Abbildung 5.3: Reinforcement Learning erweitert mittels Unsupervised Learnings.

Diese Kombination erzielt im *klassischen* Labyrinth den erwarteten Effekt und sowohl die Summe der durchschnittlich erhaltenen Belohnungen pro Episode ist deutlich gestiegen, als auch die Anzahl der benötigten Schritte pro Episode gesunken. Das Verfahren konvergiert nun bereits durchschnittlich 15 Lernepisoden früher. Sei außerdem noch ein Blick auf die Zugriffe der einzelnen Zustände geworfen.

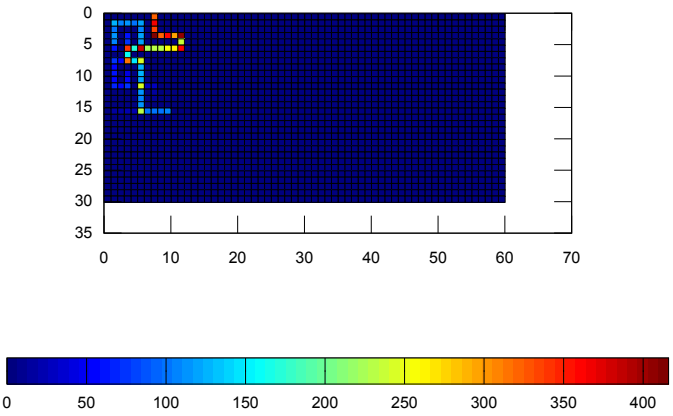

Abbildung 5.4: Zugriffsanzahl der einzelnen Zustände über 100 Episoden.

Es ist der Trend zu erkennen, dass sich der Agent zunehmend in den *Ecken* des Labyrinths aufhält, in denen Makro-Aktionen enden und folglich verändert sich das Lernverhalten des Agenten signifikant. Dies hat zur Folge, dass im späten Verlauf des Lernprozesses sich die Aufenthaltszeit des Agenten hauptsächlich auf Zustände verteilt, welche *(a)* Teil der Lösung sind und *(b)* mittels Makro-Aktionen erreicht werden und somit selbst Makro-Aktionen definiert haben. Dieses Verhalten wurde bereits zuvor von Amy McGovern und Richard S. Sutton in der Analyse *Macro-Actions in Reinforcement Learning: An Empirical Analysis* festgehalten [17]. Als Resultat ergibt sich somit eine geringere Fehlertoleranz: die Chance eine Makro-Aktion als Zufallsaktion zu wählen steigt, da die Zustände mehr Makro-Aktionen definiert haben – und diese stärker ins Gewicht fallen. Außerdem agiert der Agent durch die Makro-Aktionen nicht nur vermehrt auf solchen Zuständen, welche mittels Makro-Aktionen erreicht werden, sondern auch auf denen in der näheren Umgebung.

Für diese Zustände sind durch das Anwenden der Makro-Aktionen nur diejenigen (Zustand, Aktion)-Paare genügend exploriert, welche *innerhalb* der Makro-Aktion verwendet werden. Für die (Zustand, Aktion)-Paare, welche nicht von der Makro-Aktion abgedeckt werden, ist hingegen keine ausreichende Abschätzung über die Werte vorhanden. Dies hat zur Folge, dass (falls der Agent auf einen solchen Zustand zwischen 2 Makro-Zuständen landen) eine erneute Exploration der Umgebung außerhalb der Makro-Aktionen stattfindet und das Verfahren somit noch Schwankungen ausgesetzt ist. Um eine Stabilität der Werte dem Q-Learning ähnlich zu erreichen, wäre eine längere Lernphase von Nöten (*ca. 1200 Episoden anstelle von 1000*).
Sei nun die Kombination Q-Learning mit dem Umgebungsmodell des *Supervised Learnings* betrachtet.

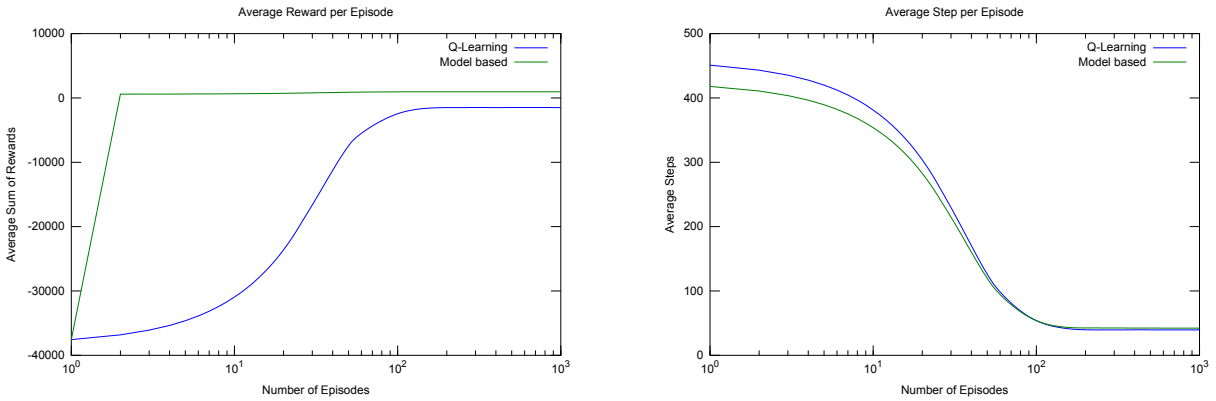

Abbildung 5.5: Reinforcement Learning erweitert mittels Supervised Learnings.

Auch diese Kombination erzielt den gewünschten Effekt: die Summe der zu erwartenden Belohnungen ist erheblich gestiegen, da die Bestrafungen mit -1000 ausfallen und lediglich die Zustandswechsel mit je -1 als Belohnung zu Buche schlagen. Ebenfalls ist die Anzahl der benötigten Schritte merkbar gesunken, allerdings fällt sie höher aus, als zuvor beim Unsupervised Learning in Abbildung 5.4. Dort benötigt der Agent bereits in der ersten Episode durchschnittlich ca. 355 Schritte, um das Ziel zu erreichen – beim Supervised Learning hingegen ca. 370 Schritte. Der Grund hierfür liegt am ϵ-Greedy Verfahren. Im Rahmen des Unsupervised Learnings können Fehlentscheidungen des ϵ-Greedy Verfahrens den Agenten zum einen gegen eine Wand, zum anderen in eine falsche Richtung führen. Im Falle der falschen Richtung können die benötigten Schritte rasant ansteigen, da per Zufall eine mehrere Schritte lange Makro-Aktion ausgewählt werden kann, was aber tatsächlich eher selten vorkommt: da jeder Zustand durchschnittlich 4 Aktionen sowie eine Makro-Aktionen besitzt, liegt die Wahrscheinlichkeit eine Makro-Aktion als Aktion zu wählen bei bereits $\frac{1}{5}$ – zusätzlich zur bereits bestehenden Wahrscheinlichkeit von nur $\epsilon = 0.1$ überhaupt eine Fehlentscheidung zu treffen.
Wird hingegen eine Mauer-Aktion gewählt, so wird nur ein Fehlschritt mit ein berechnet. Beim Supervised Learning hingegen besteht zu Beginn der Lernphase die gleiche Chance einen Schritt in die falsche Richtung, als auch in die richtige Richtung zu gehen. Im späteren Verlauf steigt hingegen die Chance, sich in die falsche Richtung zu begeben. Aus diesen Fehlentscheidungen entsteht die Diskrepanz zum Unsupervised Learning: wird eine Fehlentscheidung getroffen, so führt der Agent durchschnittlich 2 Schritte aus: einen Schritt in die falsche Richtung und einen Schritt in die richtige Richtung.
Sei nun die Kombination beider Verfahren betrachtet.
Die Graphen in der folgenden Abbildung zeigen die Erweiterungen Q-Learning mit Unsupervised Learning (*Clustering*), Q-Learning mit Supervised Learning (*Model based*) sowie die Kombination Q-Learning mit Supervised und Unsupervised Learning (*Clustering + Model based*) im Vergleich. Da hier die Y-Achse feiner ist als zuvor in Abbildung 5.4, fallen die Schwankungen des Q-Learnings im Zusammenspiel mit dem Unsupervised Learning verstärkt auf. Des Weiteren ist zu sehen, dass die Kombination des Supervised, Unsupervised sowie Reinforcement Learnings einen gesteigerten Effekt erzielt. Sowohl die Summe der erhaltenen Belohnungen steigt merkbar an und ist ca. ab der 80. Episode nahezu konstant, dicht gefolgt vom Q-Learning in Kombination mit Supervised Learning. Auch die benötigten Schritte sinken merkbar, sind aber noch immer aus den zuvor genannten Gründen höher als bei der Kombination Q-Learning mit Makro-Aktionen. Als letzte Auswertung folgt nun die zusätzliche Erweiterung mittels Verkapseln erlernten Wissens.

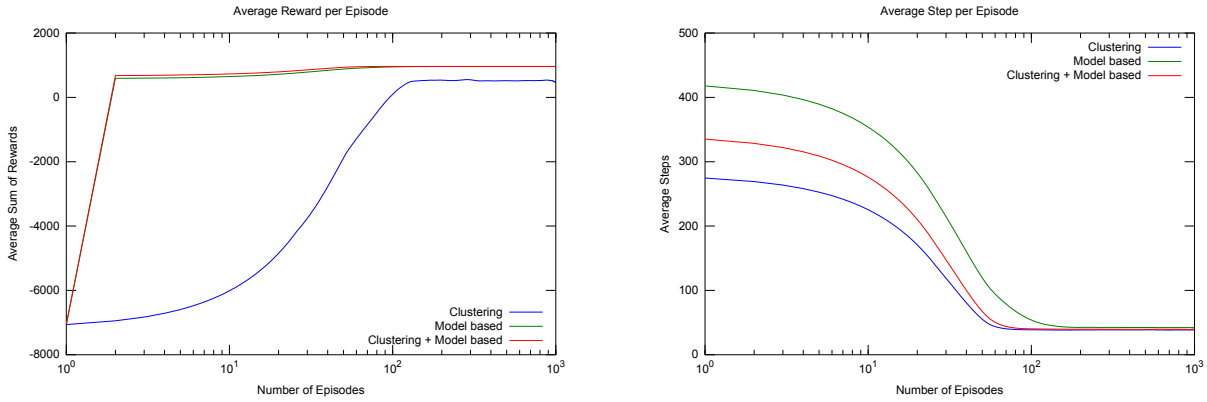

Abbildung 5.6: Reinforcement Learning erweitert mittels Supervised sowie Unsupervised Learnings.

Die folgenden Graphen zeigen die erzielten Erfolge mittels Verkapselns im Vergleich mit den zuvor erzielten Resultaten. Die im Rahmen des Verkapseln erlernten Wissens benötigten Schwellwerte x und y der Kriterien 2 und 3 aus Abschnitt 3.5.1 auf Seite 32 sind mit $x = 10$ sowie $y = 300\%$ festgelegt. Durch die Ausnutzung der engen Gänge, der Sackgassen sowie des Umgebungsmodells des Supervised Learnings ist eine nahezu sofortige Konvergenz gegen die optimale Lösung erzielt. Schon ab der 11. Episode ist hier die Lernphase beendet und der optimale Pfad von Start nach Ziel gefunden. Die Bezeichnung *Encaps* ist eine Abkürzung des Wortes *Encapsulating* (englisch für Verkapseln) und beschreibt somit in den folgenden Graphen die Erweiterung mittels Verkapseln erlernten Wissens.

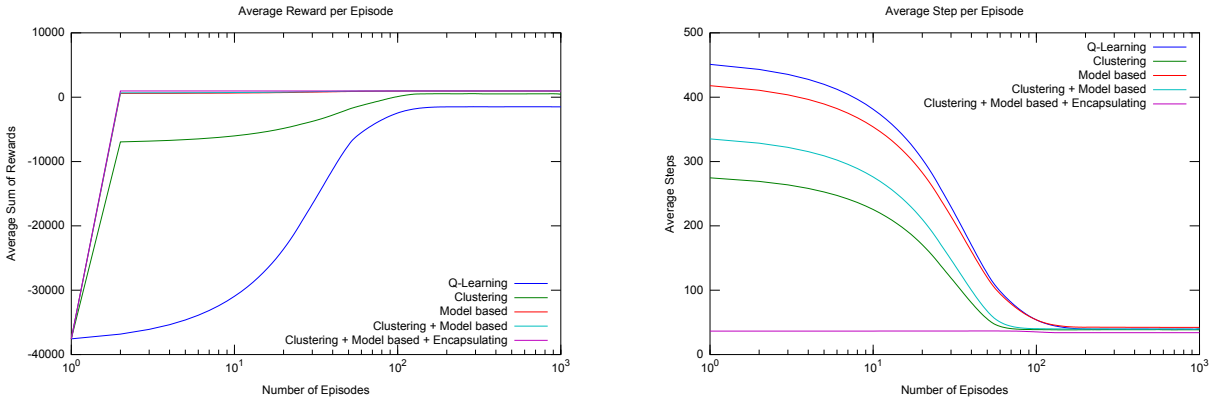

Abbildung 5.7: Zusätzliche Erweiterung durch Verkapseln erlernten Wissens. Schwellwerte x und y sind hierbei mit $x = 10$ und $y = 300\%$ gewählt.

Aus den erhobenen Daten bilden sich zum Schluss die folgenden Zahlen für die Werte *Median* sowie *durchschnittliche Belohnung pro Schritt*:

klassische Labyrinth	RL	UL+RL	SL+RL	UL+SL+RL	UL+SL+RL+Enc.
Median	-2427.52	221.94	947.87	956.97	966.67
Belohnung pro Schritt	-47.09	5.16	17.7	20.46	28.01

Die hier verwendeten Abkürzungen *RL*, *UL*, *SL* sowie *Enc* stehen für die verwendeten Verfahren: **R**einforcement **L**earning entspricht der reinen Anwendung des Q-Learnings, **U**nsupervised **L**earning der Eweiterung mittels Makro-Aktionen und des Clusterings, **S**upervised **L**earning entspricht der Erweiterung mittels Umgebungsmodell und **Enc**apsulating entspricht zu guter Letzt dem Verkapseln erlernten Wissens.

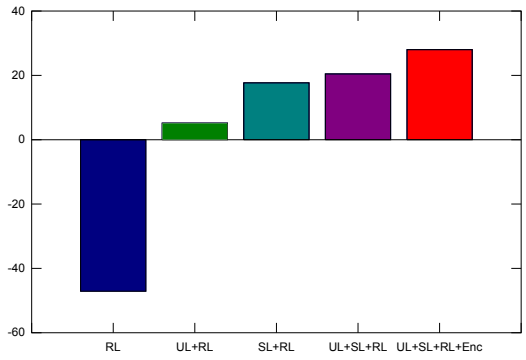

Abbildung 5.8: Durchschnittliche Belohnung pro Schritt aller vorgestellten Kombinationen veranschaulicht.

5.2 freie und offene Labyrinthe

Da sich diese beiden Arten der vorgestellten Labyrinthe sehr stark ähneln und sich somit die Auswertungen der Erwartung nach ebenfalls stark ähneln, werden im Folgenden beide Labyrinth-Arten zusammengefasst betrachtet. Hierbei wird Abwechselnd zu jeder Stufe der verwendeten Architektur zunächst das *freie* Labyrinth ausgewertet und anschließend das *freie* Labyrinth.

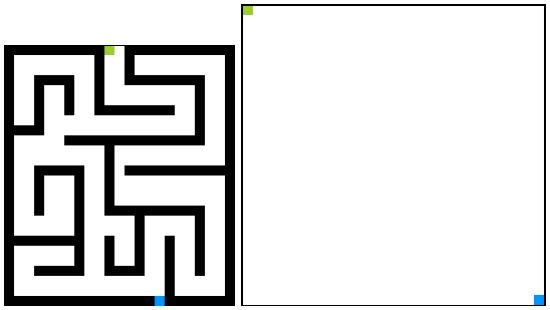

Abbildung 5.9: Die untersuchten Labyrinthe aus den Familien der *freien* sowie *offenen* Labyrinthe.

Erwartungshaltung

Da sowohl *freie* als auch *offene* Labyrinthe aus beliebig breiten Gängen bestehen, werden die hier vorgestellten Verfahren einen geringen Erfolg erzielen. Da die Gänge breiter als einen Zustand sind, wird das Clustering aus Sicherheitsgründen keine Makro-Aktionen definieren, welche das Verfahren beschleunigen könnten. Da somit das Verfahren allerdings das dem reinen Q-Learnings entspricht, sollte die Kombination Unsupervised mit Makro-Q-Learning auf keinen Fall schlechter abschneiden, als Q-Learning selbst. Mittels der Erweiterung durch Supervised Learnings ist die Erwartungshaltung zweigeteilt: in *freien* Labyrinthen sollte eine Steigerung der Konvergenz eintreten, da es ungültige (Zustand, Aktion)-Paare gibt, welche durch das Umgebungsmodell gefiltert werden können. In *offenen* Labyrinthen hingegen sollte das Umgebungsmodell keinen Vorteil bringen, da es entweder eine sehr niedrige Anzahl oder aber gar keine Mauern und somit ungültige (Zustand, Aktion)-Paare gibt. Daher sollte auch diese Erweiterung dem Q-Learning ähnlich gegen die optimale Lösung konvergieren und keine nennenswerte Beschleunigung der Konvergenz erzielen.

Sei zunächst wieder die reine Anwendung des Reinforcement-Learnings betrachtet, um die erzielten Ergebnisse als Referenzwerte festzuhalten. Als erstes wird das *freie* Labyrinth betrachtet.
Aus diesen Resultaten lässt sich eine Konvergenz des Q-Learnings ungefähr ab der 500. Episode betrachten und es ist ebenfalls eine gute Mischung aus Exploration sowie Exploitation zu sehen, da bereits nach 150 Episoden alle potenzielle Lösungen gefunden wurden, worunter auch der kürzeste Weg zu finden ist. Auch hier wurde der Weg rechts vom Ziel früh verworfen (*ab der 5. Episode*).

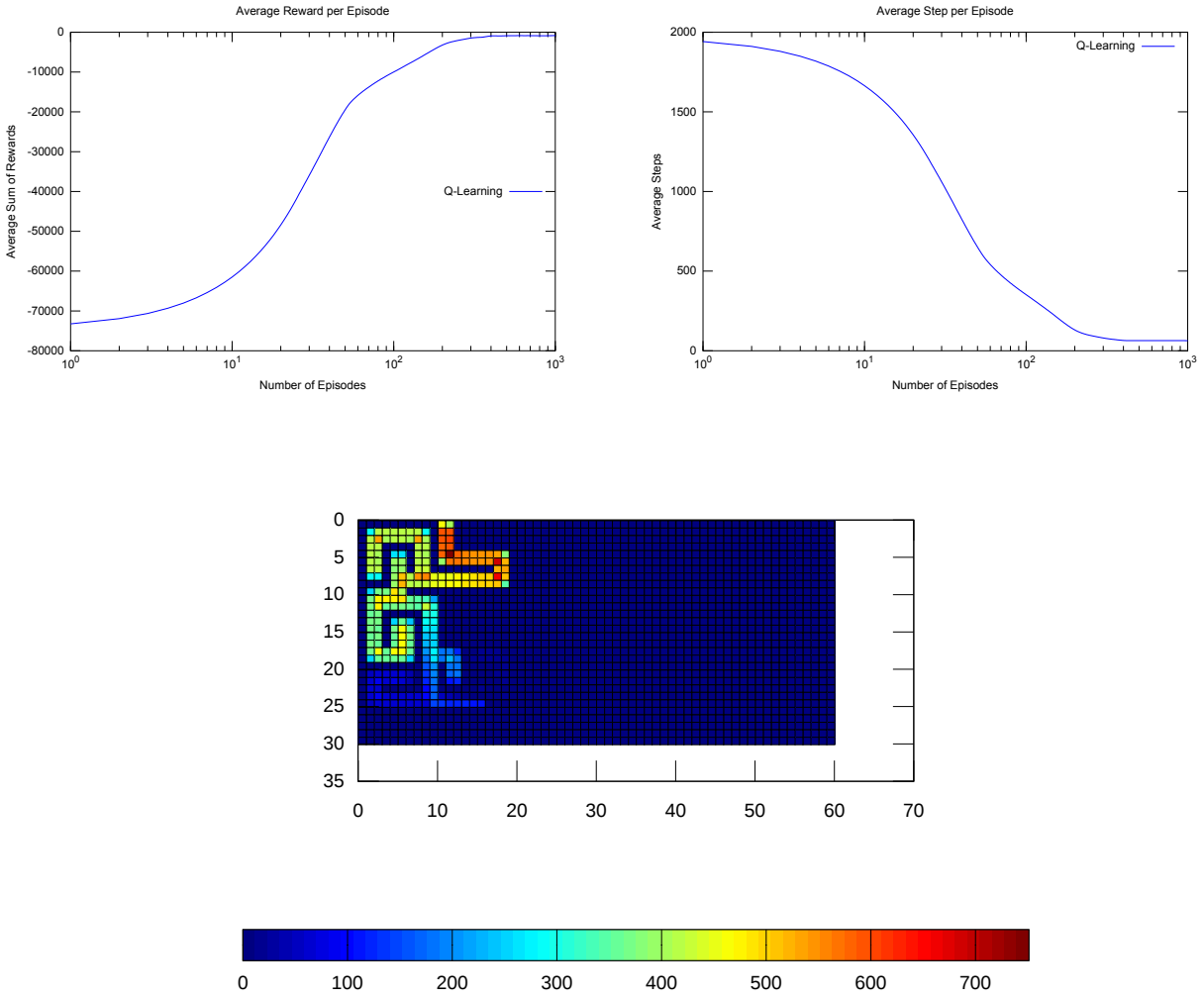

Abbildung 5.10: Daten des Reinforcement Learnings auf das *freie* Labyrinth.

Seien nun die Referenzwerte des *offenen* Labyrinths erhoben.

Die Daten dieser Labyrinth-Art wurden über 4.000 Lernepisoden erhoben und als Semi-Log-Plot an der X Achse gezeichnet. Da die Daten nach den 4.000 Lernepisoden keine Veränderungen widerfahren, wurde auf die Weiterführung der Daten verzichtet.
Hier wurde ungefähr ab der 2100. Episode die optimale Lösung gefunden und eine Konvergenz erreicht. Aus der Zugriffsanzahl der einzelnen Zustände, welche über 100 Episoden betrachtet wurde, lässt sich eine zufriedenstellende Verteilung der Häufigkeit feststellen. Außerdem wurden bereits 2 mögliche Lösungsansätze gefunden, sowie die Zustände rechts vom Ziel verworfen.

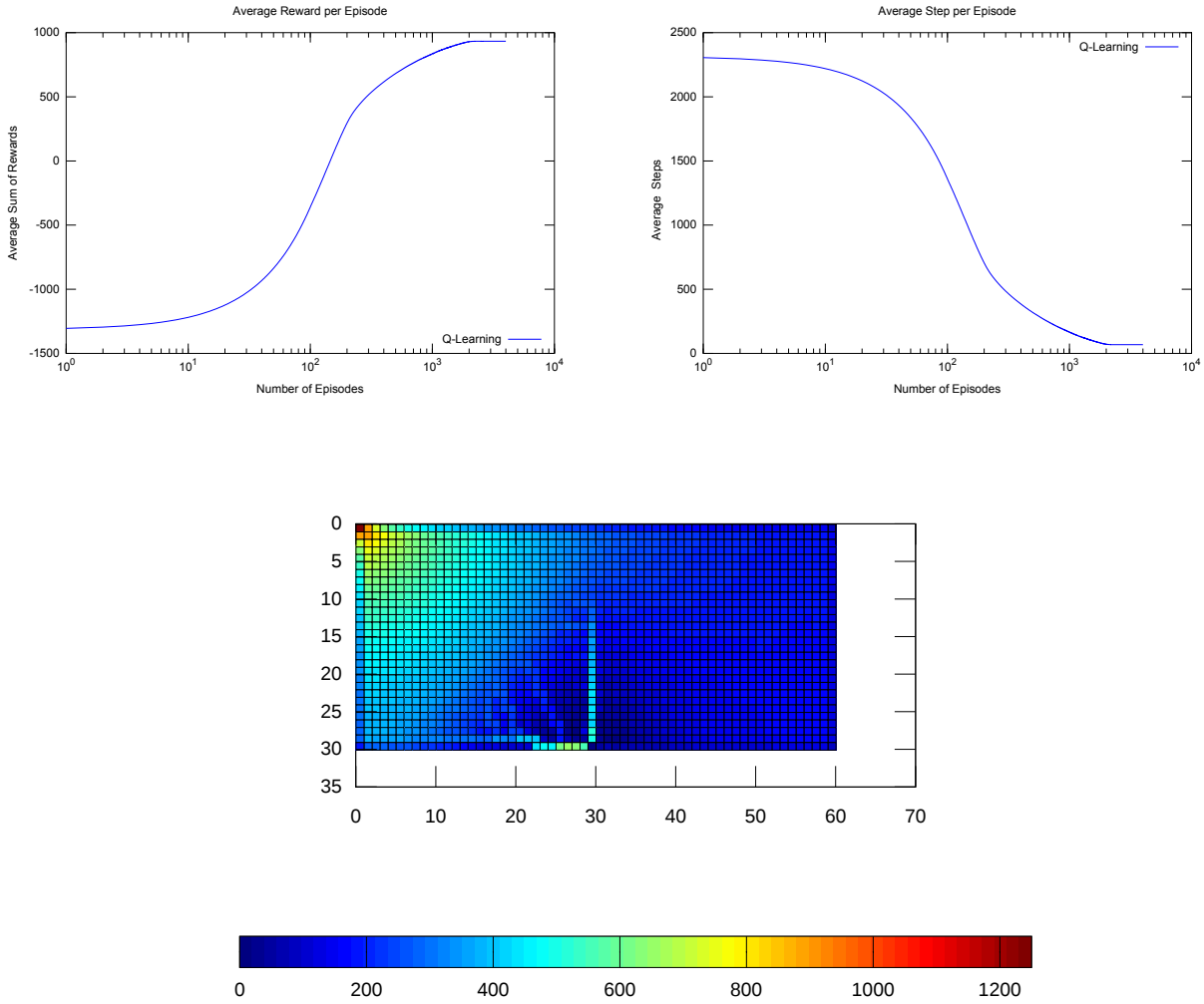

Abbildung 5.11: Daten des Reinforcement Learnings auf das *offene* Labyrinth.

Nachdem nun die Referenzwerte *freis Labyrinth: ca. 500 Episoden* sowie *offenes Labyrinth: ca. 2100 Episoden* erhoben wurden, sei sich nun der ersten Erweiterung zugewendet: dem Unsupervised Learning.

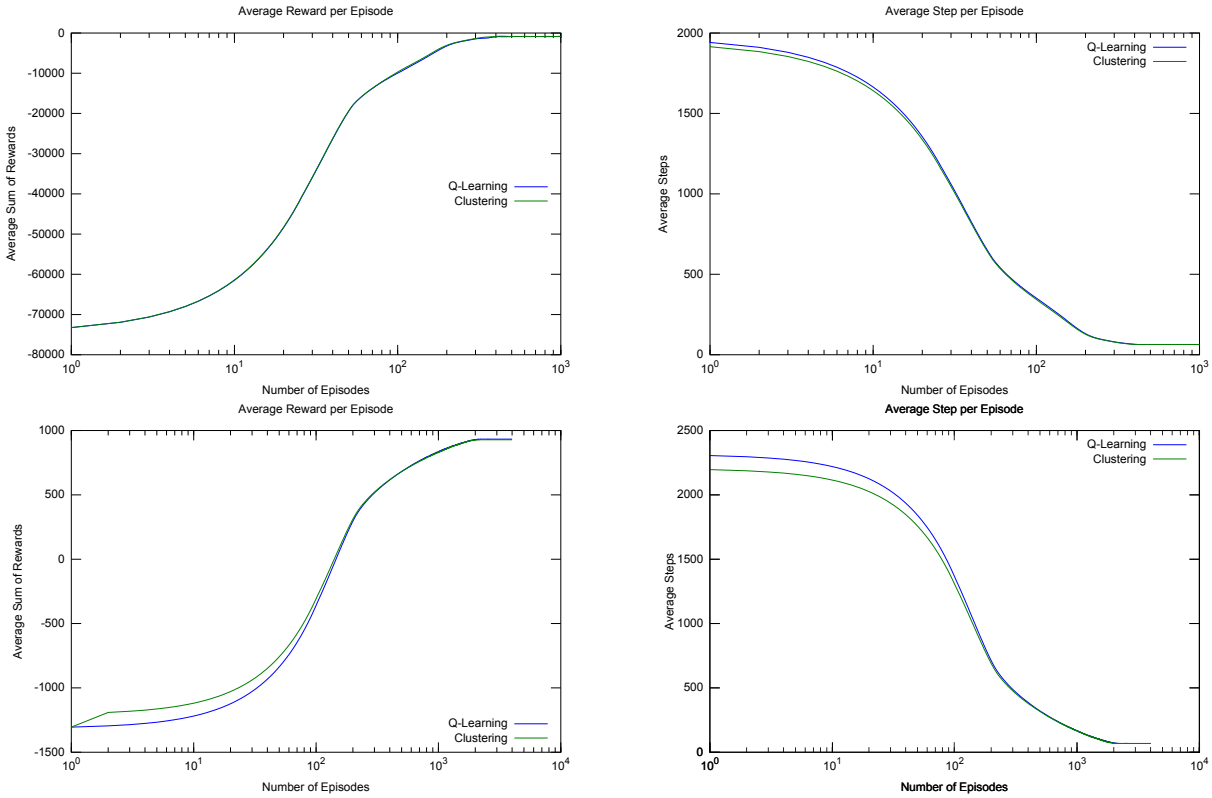

Abbildung 5.12: Reinforcement Learning erweitert mittels Unsupervised Learning.

Die in dieser Kombination erzielten Ergebnisse bestätigen die vorhergegangene Erwartungshaltung: in der ersten Zeile sind die Graphen des *freien* Labyrinths zu sehen, in der zweiten hingegen die des *offenen* Labyrinths. In beiden Fällen fand eine Verbesserung im Rahmen der durchschnittlich benötigten Schritte statt, welche sich durch die Art und Weise des Clusterings erklären lässt. Agiert der Agent auf einem zuvor unbekannten Labyrinth, so folgt er zunächst einige Schritte weit einer Richtung. Findet in dieser Richtung kein Richtungswechsel statt, so interpretiert das Unsupervised Learning in der ersten Clustering-Phase diese Daten als Gänge und definiert entsprechende Makro-Aktionen. Werden diese Makro-Aktionen ausgeführt, so erhalten mehrere Q-Werte neue Abschätzungen und es werden weniger Schritte zur weiteren Exploration benötigt. Dies hat allerdings keine nennenswerte Verbesserung der durchschnittlich erhaltenen Summe von Belohnungen pro Episode zu folge, wie in den Graphen auf der linken Seite zu sehen ist. Im Falle des *freien* Labyrinths verhält sich die Kombination somit analog zum Q-Learning. Hingegen im Falle des *offenen* Labyrinths findet zu Beginn der Lernphase eine Verbesserung statt, welche im weiteren Verlauf der Exploration des Labyrinths verschwindet.
Sei nun die Erweiterung mittels Supervised Learnings betrachtet, welche im Falle des *freien* Labyrinths vielversprechender erscheint.

Die folgende Abbildung 5.13 zeigt die erhobenen Daten der Kombination *Q-Learning mit Umgebungsmodell* im Vergleich zum reinen *Q-Learning* in der gleichen Reihenfolge wie zuvor: die erste Zeile enthält die Daten des *freien* Labyrinths, die zweite Zeile hingegen die des *offenen* Labyrinths. Auch hier wurde die Erwartungshaltung eingehalten: die Summe der durchschnittlich erhaltenen Belohnungen des freien Labyrinths ist *angestiegen*, die des offenen Labyrinths hingegen *gleich geblieben*. Diese Auswirkungen lassen sich ebenfalls in den rechten Graphen verdeutlichen: im Rahmen des freien Labyrinths wurden zwar ungültige (Zustand, Aktion)-Paare herausgefiltert, doch es bleibt noch immer eine große Anzahl an (Zustand, Aktion)-Paaren übrig, welche es zu explorieren gilt. Folglich ist eine geringe Verbesserung der Konvergenz zu verzeichnen, welche zusammen mit der Anzahl der Mauern im Labyrinth ansteigt. Im hier untersuchten offenen Labyrinth hingegen existieren keine Wände und somit keine ungültigen (Zustand, Aktion)-Paare, welche die Konvergenz beeinträchtigen könnten. Folglich verhält sich das Q-Learning *mit* Umgebungsmodell korrekterweise analog zum Q-Learning *ohne* Umgebungsmodell.

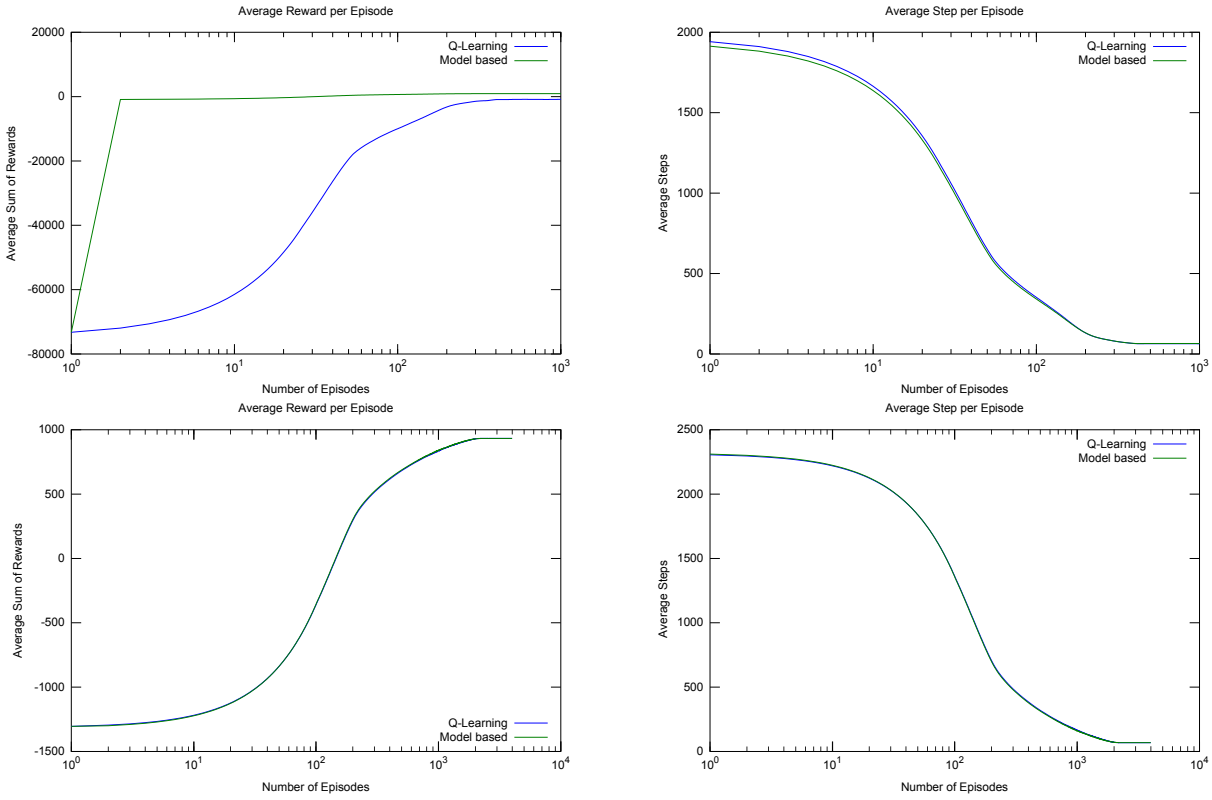

Abbildung 5.13: Reinforcement Learning erweitert mittels Supervised Learning.

Nachdem nun die einzelnen Erweiterungen betrachtet wurden, wird sich nun der Kombination gewidmet. Die Ergebnisse der Kombination Q-Learning mit Unsupervised sowie Supervised Learning (*Clustering + Model based* in der Graphik) werden erneut im direkten Vergleich mit den einzelnen Erweiterungen in Abbildung 5.14 aufgezeigt. Da in einer Kombination aus Umgebungsmodell und Clustering das Clustering von Beginn an *vollständig* und somit korrekt ist, werden genau dann Makro-Aktionen definiert, wenn zwei Transitknoten mindestens einen Zustand weit entfernt sind. Andernfalls entsprächen diese Makro-Aktionen, welche zwei direkt benachbarte Knoten miteinander verbinden, den einfachen Aktionen und würden keinen Effekt erzielen. Da in *freien* als auch in *offenen* Labyrinthen hingegen die Gänge der Definition aus Abschnitt 4 nach beliebig breit sein dürfen, sind mit dem hier verwendeten Verfahren keine bis hin zu einer geringen Anzahl an Makro-Aktionen definierbar. In den konkret verwendeten Labyrinthen sind diese zur Definition von Makro-Aktionen benötigten 1-Zustand breiten Gänge nicht vorhanden und es sind somit keine Makro-Aktionen definierbar. Dies hat zur Folge, dass es keine Beschleunigung der Konvergenz gibt und das Verfahren somit identisch zum reinen *Q-Learning mit Umgebungsmodell* ist.

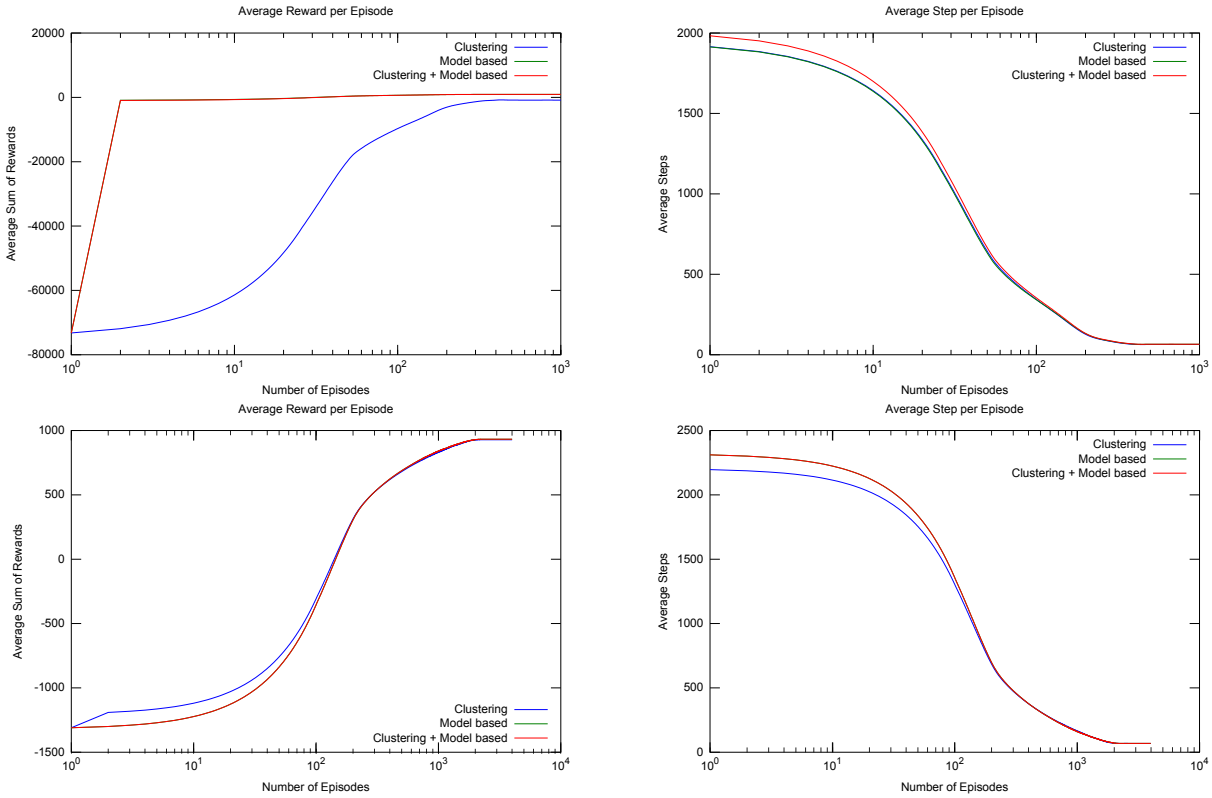

Abbildung 5.14: Reinforcement Learning erweitert mittels Kombination des Unsupervised und Supervised Learning.

Als letzte Erweiterung bleibt nun noch die *Verkapselung erlernten Wissens* auszuwerten. Die durch diese Erweiterung erzielten Ergebnisse sind wieder im direkten Vergleich mit allen zuvor beschriebenen Kombinationen aufgezeigt. Dieser Vergleich ist in Abbildung 5.15 sowie 5.16 zu sehen. Da, wie zuvor bereits analysiert, das Umgebungsmodell mit Hilfe des Clusterings in beiden Labyrinthen keine Makro-Aktionen definiert, ist auch diese Erweiterung den Erwartungen nach keine erhebliche Verbesserung. Lediglich das Verkapseln erzielt aufgrund der hohen Iterationen mit den Werten $x = 100$ sowie $y = 300\%$ im späteren Verlauf der Lernphase.

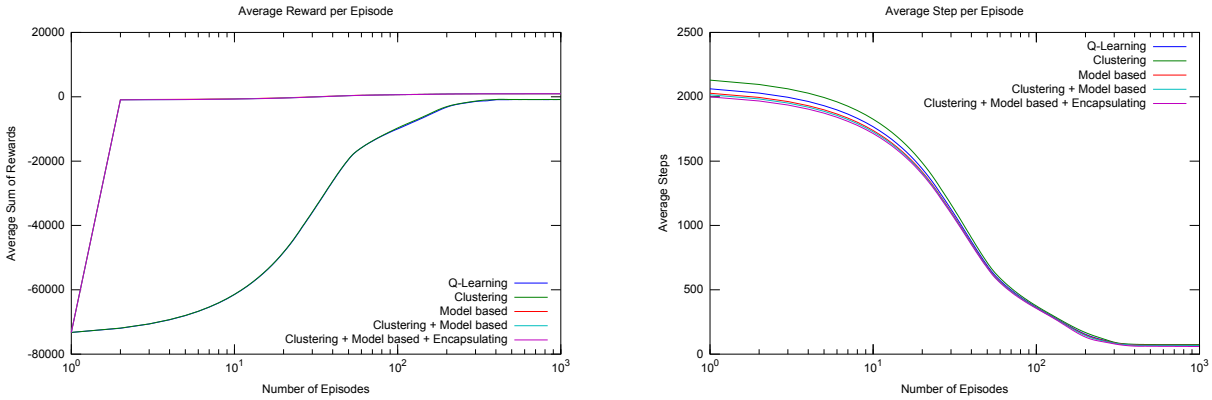

Abbildung 5.15: Zusätzliche Erweiterung der Kombination mit Verkapselung des *freien* Labyrinths.

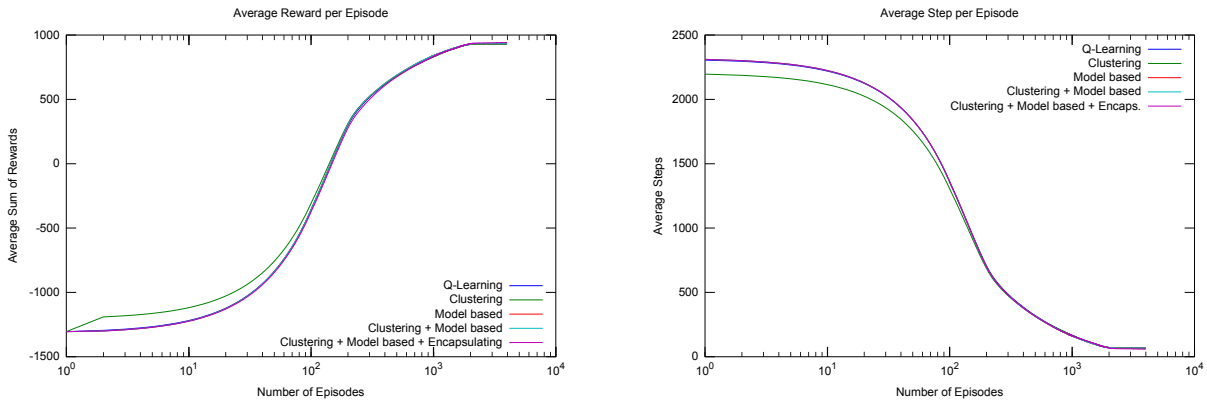

Abbildung 5.16: Zusätzliche Erweiterung der Kombination mit Verkapselung des *offenen* Labyrinths.

Aus den erhobenen Daten bilden sich zum Schluss die folgenden Zahlen für die Werte *Median* sowie *durchschnittliche Belohnung pro Schritt* für das *freie* Labyrinth:

freies Labyrinth	RL	UL+RL	SL+RL	UL+SL+RL	UL+SL+RL+Enc.
Median	-3985.28	-3905	847	849	851
Belohnung pro Schritt	-25.48	-25.03	5.29	5.305	5.39

Sowie für das *offene* Labyrinth:

offenes Labyrinth	RL	UL+RL	SL+RL	UL+SL+RL	UL+SL+RL+Enc.
Median	815.85	815.04	816.39	816.37	819.19
Belohnung pro Schritt	4.36	4.6	4.66	4.65	4.72

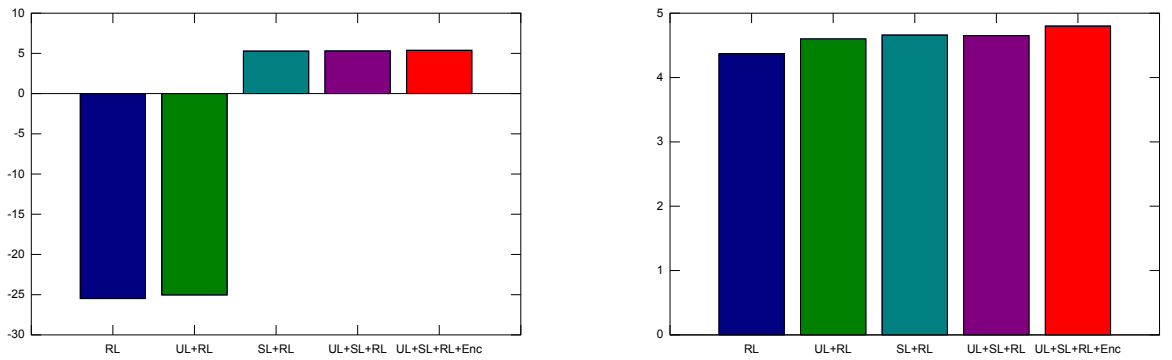

Abbildung 5.17: Durchschnittliche Belohnung pro Schritt aller vorgestellten Kombinationen veranschaulicht.

Abschließend lässt sich im Rahmen dieser beiden Labyrinth-Arten festhalten, dass das verwendete Unsupervised Verfahren zu Beginn eine Beschleunigen der Konvergenz erzielen kann, welche im späteren Verlauf allerdings nachlässt. Das Umgebungsmodell des Supervised Learnings erreicht im Falle des *freien* Labyrinths hingegen einen anhaltenden Effekt. Eine Kombination beider entspricht hierbei annähernd dem gleichen Verfahren, wie es zuvor die Kombination des Q-Learnings mit Umgebungsmodell darstellte und der daraus erzielte Nutzen hängt von der Anzahl der 1-Zustand breiten Gänge ab. Als weitere Verbesserung in diesen Labyrinthen stellte sich das Verkapseln erlernten Wissens heraus, da zum Ende der Lernphase eine steigende Anzahl von Standardaktionen definiert wird.

5.3 knifflige Labyrinthe

Als letzte Labyrinth-Art im Rahmen dieses Experiments wird das *knifflige* Labyrinth untersucht. Diese Labyrinth-Art stellt eine Kombination der zuvor beschriebenen Labyrinth-Arten dar: es gibt sowohl 1-Zustand als auch n-Zustand breite Gänge, durch die sich der Agent zu navigieren hat. Außerdem existieren Sackgassen sowie Areale, in denen mehrere Abwandlungen des optimalen Wegs zu finden sind. Als weiteres Merkmal sei bei dem hier verwendeten Labyrinth die Länge angemerkt: der optimale Weg vom Start zum Ziel beträgt 602 Schritte.

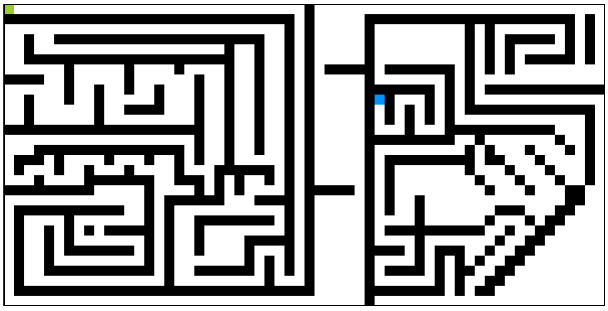

Abbildung 5.18: Das untersuchte Labyrinth aus der Familie der *kniffligen* Labyrinthe.

Erwartungshaltung

Die Erwartungshaltung in dem hier verwendeten Labyrinth ist zweigeteilt: zum einen sollten sowohl das vorgestellte Unsupervised sowie Supervised Verfahren als auch das Verkapseln erlernten Wissens Verbesserungen der Konvergenz erzielen, zum anderen könnte das Problem des *temporal credit assignments* auftreten. Der Begriff des *temporal credit assignments* beschreibt die Problemsituation, welche entsteht wenn ein Endzustand nach einer Sequenz von Aktionen erreicht wird und eine entsprechende Belohnung erhalten wird. In dieser Situation ist die Frage, welche der in der Sequenz verwendeten Aktionen tatsächlich für die Belohnung verantwortlich ist. In den Fällen zuvor konnte dies mittels der geringen Entfernung und genügend Iterationen vorausgesetzt nachvollzogen werden. Im Rahmen dieses Labyrinths hingegen könnte dies zum Problem werden, da eine 600 Schritte entfernte (*discounted*) Belohnung bereits nur noch $0.98^{600} r_{t+600} = (5.44058269 * 10^{-6}) r_{t+600}$ entspräche.

Resultate

Dem Schema der beiden vorangegangenen Abschnitte folgend wird zunächst das reine Q-Learning ohne Erweiterungen auf das Labyrinth angewendet, um die Referenzwerte für die folgenden Vergleiche zu ermitteln.

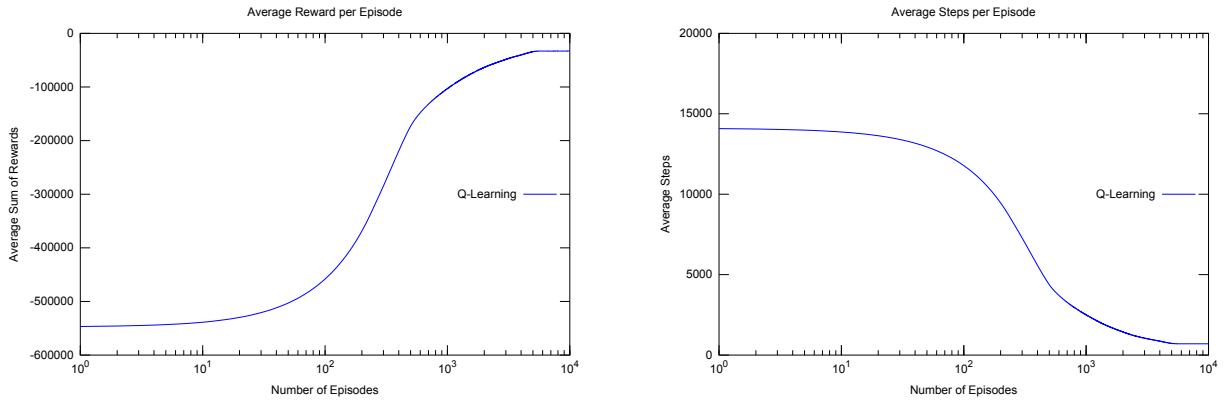

Abbildung 5.19: Daten des Reinforcement Learnings auf das *knifflige* Labyrinth.

Aufgrund der Länge des Labyrinths findet den Erwartungen nach die Konvergenz des Labyrinths vergleichsweise schleppend statt, da das Q-Learning den (Zustand, Aktion)-Paaren, welche dem Zielzustand näher sind, früh eine bessere Abschätzungen liefert und die zum ziel führenden (Zustand, Aktion)-Paare somit vom Ziel- zum Startzustand iterativ verfeinert werden. Aus diesem Grund ist in der Anfangsphase des Q-Learnings in der Nähe des Startzustands noch nicht gänzlich klar, welche (Zustand, Aktion)-Paare tatsächlich zum Ziel führen und welche nicht (*siehe vorangegangene Beschreibung des temporal credit assignment Problems*). Eine Konvergenz findet mittels reinem Q-Learnings ca. ab der 8500. Episode statt.
Ein Blick auf die Anzahl der Zugriffe eines jeden Zustands über 1000 Episoden offenbart im Anfangsbereich des Labyrinths viele Zustände, in denen die Zugriffsanzahl besonders hoch ist.

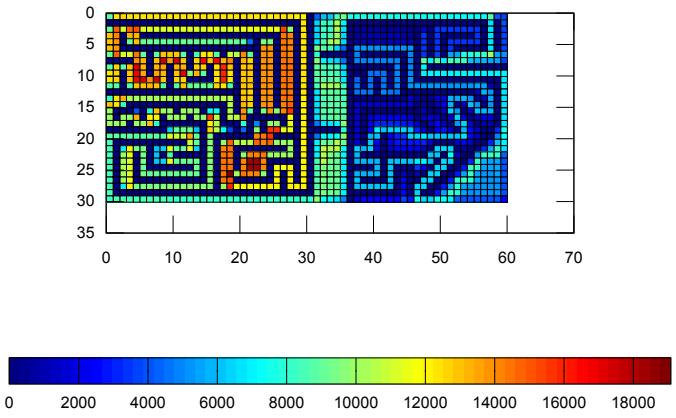

Abbildung 5.20: Zugriffsanzahl der einzelnen Zustände über 1000 Episoden.

Der Grund hierfür liegt ebenfalls an dem zuvor besprochenen Zustand innerhalb der Anfangsphasen des Lernprozesses. Besonders in den roten Zuständen ist eine eindeutige Richtung noch nicht gegeben und die erhaltene Belohnung des Endzustands ist noch nicht bis zu den dazugehörigen (Zustand, Aktion)-Paaren durchgedrungen.
Als nächster Schritt sei die Erweiterung mittels Unsupervised Learning analysiert.

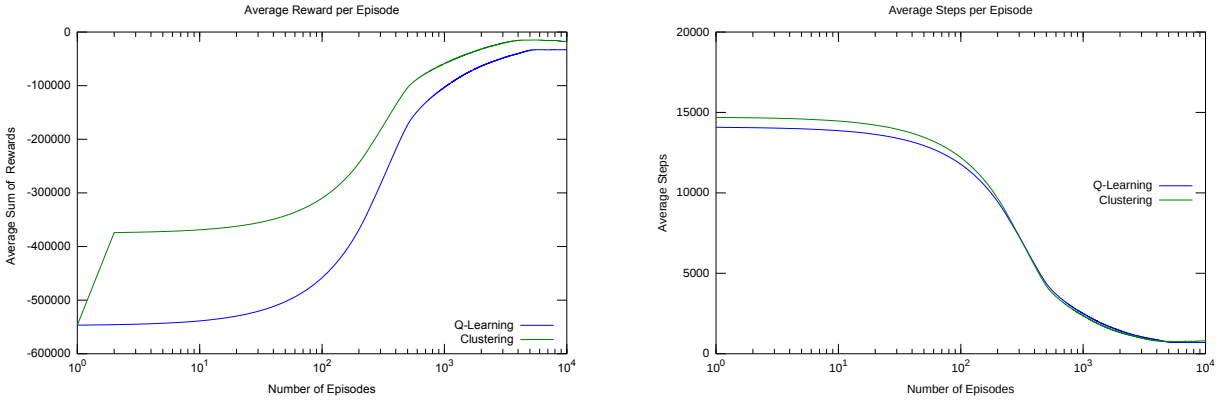

Abbildung 5.21: Reinforcement Learning erweitert mittels Unsupervised Learning.

Aufgrund der vielen engen Gänge ist es in diesem Labyrinth möglich eine Vielzahl von Makro-Aktionen zu definieren, welche anschließend die Konvergenz beschleunigen. Außerdem ist am Ende des linken Graphens erneut die bereits zuvor beschriebene Instabilität der Werte zu betrachten. Diese entsteht in diesem Labyrinth erneut dadurch, dass sich die Aufenthaltszeit des Agenten in Transitzuständen erhöht und weitere (Zustand, Aktion)-Paare exploriert werden, welche zuvor durch die Makro-Aktionen nicht abgedeckt wurden. Die Anzahl der benötigten Schritte zum Ziel steigt zunächst aufgrund der zusätzlichen Verwendung der Makro-Aktionen neben den normalen Aktionen an, verringert sich aber über den Verlauf der Lernphase.
Es folgt nun die Erweiterung mittels Supervised Learning betrachtet.

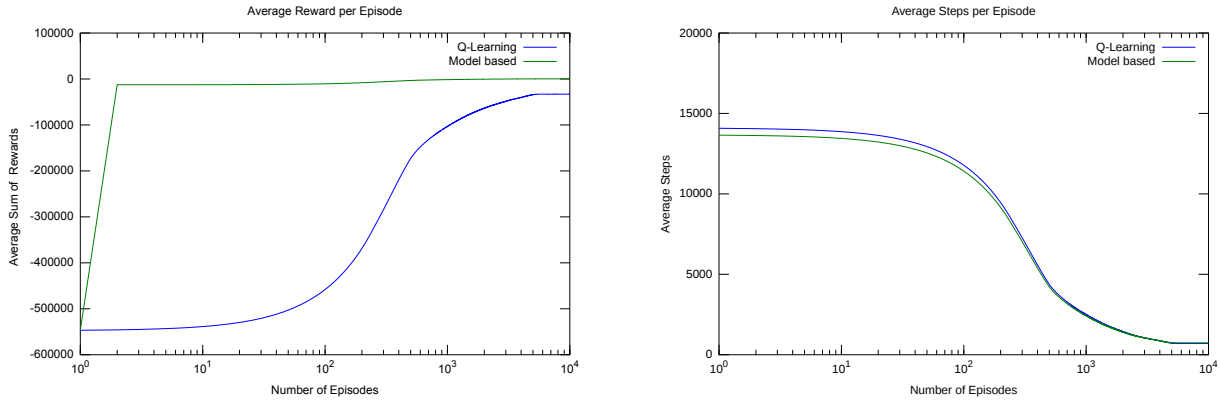

Abbildung 5.22: Reinforcement Learning erweitert mittels Supervised Learning.

Auch hier ist durch das herausfiltern der (Zustand, Aktion)-Paare, welche eine Bestrafung mit sich führen, die Summe der erhaltenen Belohnungen auf einem höheren Niveau, wodurch auch die Anzahl der durchschnittlich benötigten Schritte zum Ziel sinkt, da die Exploration auf den potenziell zur Lösung gehörenden Zuständen zu Beginn höher ist, als beim reinen Q-Learnings. Folgende Abbildung betrachtet hierbei das Lernverhalten der Kombination Q-Learning mit Umgebungsmodell im Detail.

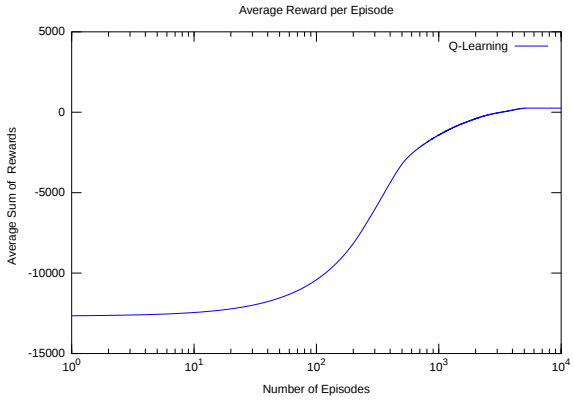

Abbildung 5.23: Reinforcement Learning erweitert mittels Supervised Learning im Detail.

Mittels dieser Erweiterung wird eine Konvergenz gegen die optimale Lösung bereits ungefähr 300 Episoden früher gefunden, als es beim reinen Q-Learning der Fall ist. Es bleibt nun abzuwarten, wie die Kombination beider Verfahren die Konvergenz beeinflusst.
Hierzu sei nun die Kombination beider Verfahren im direkten Vergleich zur Kombination *Q-Learning mit Umgebungsmodell* betrachtet.
Da das Makro-Q-Learning die Exploration des Agenten derart beeinflusst, dass sich der Agent größtenteils auf Transitknoten aufhält, findet eine besonders starke Exploration dieser wie in den zuvor analysierten Labyrinthen statt. Außerdem begünstigt die Erweiterung mittels Umgebungsmodell diese Aufenthaltszeit, da negative Aktionen herausgefiltert wurden und somit auf den Transitknoten ein erhöhter Fokus liegt. In dem untersuchten Labyrinth dieser Kategorie nehmen Makro-Aktionen eine signifikante Stellung ein, da sie im Vergleich zu den zuvor betrachteten Labyrinthen eine erhöhte Anzahl von Schritten ausführen und somit die Exploration dieser Aktionen einen bemerkbaren Mehraufwand zur Folge haben. Dieser Mehraufwand ist im rechten Graphen der Abbildung 5.24 zu sehen: eine Kombination aus Umgebungsmodell sowie Clustering hat im Kontext der durchschnittlich benötigten Schritte vom Start- zum Zielzustand eine signifikant schlechtere Performanz, als beispielsweise das Clustering oder Umgebungsmodell allein.

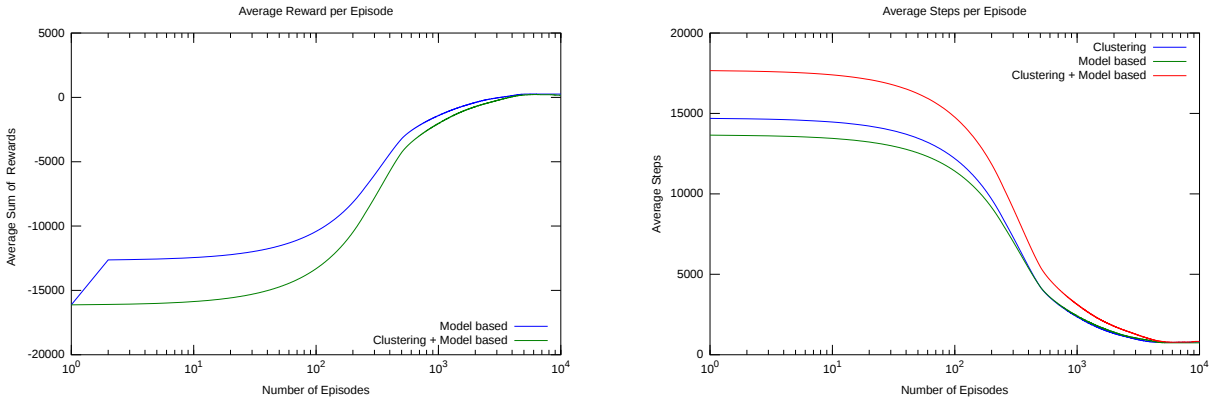

Abbildung 5.24: Reinforcement Learning erweitert mittels Unsupervised sowie Supervised Learning.

Sei nun zum Abschluss der Analyse diese Labyrinths die finale Erweiterung mittels Verkapseln betrachtet. Die hier verwendeten Parameter sind aufgrund der hohen Anzahl von Iterationen auf $x = 1200$ sowie $y = 500\%$ gesetzt.

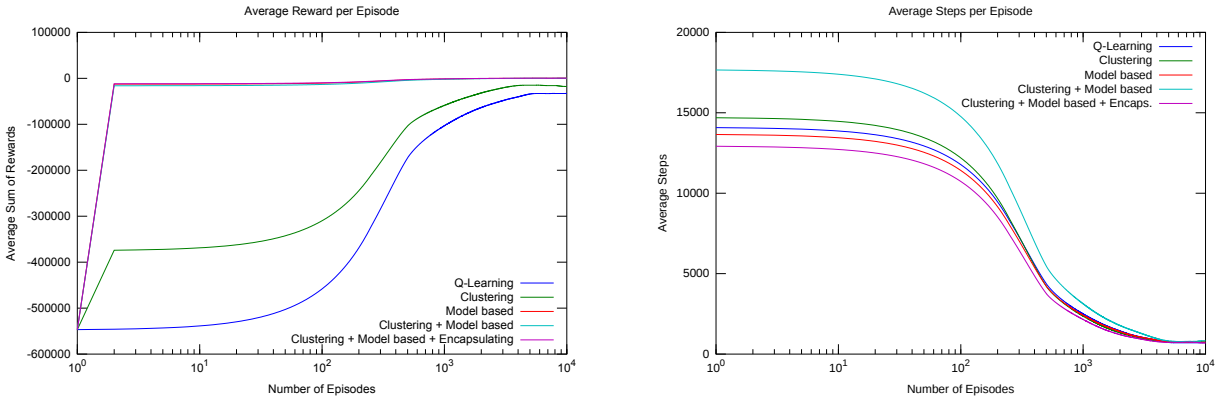

Abbildung 5.25: Zusätzliche Erweiterung der Kombination mit Verkapselung.

Die Ergebnisse, welche mittels dieser Kombinationen erzielt wurden, verdeutlichen den Nutzen des Verkapselns und der Sackgassen enorm: die Anzahl der benötigten Schritte zum Ziel sinkt signifikant: in der Anfangsphase des Lernprozesses benötigt die Kombination *Q-Learning mit Umgebungsmodell, Clustering sowie Verkapeln* durchschnittlich ca. 1200 Schritte weniger als die Kombination aus *Q-Learning und Umgebungsmodell* und sogar ca. 1500 Schritte weniger als das Q-Learning allein. Eine Konvergenz fand durchschnittlich ab der 7400. Episode statt — 1100 Episoden früher als das reine Q-Learning, ca. 800 Episoden früher als Q-Learning mit Umgebungsmodell.

Aus der letzten Analyse bilden sich zum Schluss die folgenden Zahlen für die Werte *Median* sowie *durchschnittliche Belohnung pro Schritt*:

knifflige Labyrinth	RL	UL+RL	SL+RL	UL+SL+RL	UL+SL+RL+Enc.
Median	–59294.3	-31472	-365	-618.51	-254.77
Belohnung pro Schritt	–43.22	-22.87	-0.2075	-0.3752	-0.249

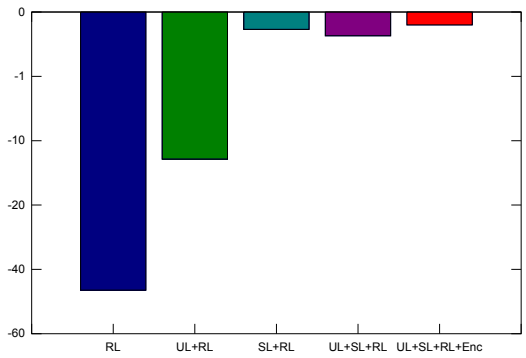

Abbildung 5.26: Durchschnittliche Belohnung pro Schritt aller vorgestellten Kombinationen veranschaulicht.

6 Zusammenfassung

Das Gehirn ist noch längst nicht in seine Gänze erforscht und wird durch die Bemühungen der Psychologie, Neurologie sowie der Informatik in den vergangenen Jahren erst Ansatzweise verstanden. Eine komplexe und wesentliche Aufgabe des Gehirns stellt das *Lernen* dar – doch wie funktioniert das menschliche Lernen genau? Forschungen der vergangenen Jahre deckten auf, dass das Gehirn beim Lernen verschiedene Lernparadigmen und somit -algorithmen verwendet, welche sich gegenseitig unterstützen [10]. Doch ist dies auch auf maschineller Ebene beim *maschinellen Lernen* möglich? Um diese Frage zu beantworten wurden zunächst in Kapitel 2 die derzeitigen Grundlagen des menschlichen sowie maschinellen Lernens skizziert und anschließend mögliche Architekturen vorgestellt, welche versuchen die Kombination verschiedener Lernparadigmen zu ermöglichen. Anschließend wurden die im Rahmen der Untersuchungen verwendeten Verfahren in den Bereichen Reinforcement, Unsupervised sowie Supervised Learning vorgestellt, sowie eine weitere Erweiterung um das Problemszenario zu vereinfachen erläutert. Hierbei wurde das Q-Learning als Basis zur Problemlösung gewählt, da es zum einen nachgewiesen Konvergiert und zum anderen im Rahmen des hier verwendeten Problemszenarios zur Lösung prädestiniert sowie intuitiv ist [29], [30].
Im Rahmen des Unsupervised Learnings wurde ein Verfahren gewählt, welches zunächst zur Lösung relevante Zustände auffindet, diese clustert und anschließend Makro-Aktionen auf ihnen definiert, um die Exploration des Labyrinths auf diese Zustände zu verstärken [17]. Als Algorithmus des Lernparadigmas Supervised Learnings wurde anschließend in Abschnitt 3.4 ein Verfahren erstellt, welches ein Umgebungsmodell des Problemszenarios erzeugt und anschließend die Aktionsmenge der zu untersuchenden Zustände auf gültige (Zustand, Aktion)-Paare beschränkt.

Mittels dieser Verfahren wurde iterativ eine Architektur entwickelt und auf die verschiedenen Labyrinth-Arten aus Kapitel 4 angewendet. Anschließend wurden die verschiedenen Kombinationen auf diese Labyrinthe angewendet und in Kapitel 5 jeweils für sich betrachtet ausgewertet. Die hier erzielten Resultate verdeutlichen, dass eine solche Kombination verschiedener Lernparadigmen ein weiter zu untersuchendes Potential aufdecken. So fand in diesem Experiment anhand einer schrittweisen Erweiterung der Hauptkomponente des Reinforcement Learnings (*Basalganglien*) durch die Unterstützung des Supervised Learnings (*Kleinhirn*) sowie des Unsupervised Learnings (*Großhirnrinde*) in einigen Fällen eine bemerkbare Beschleunigung der Konvergenz statt. Anhand dieser Ergebnisse lässt sich somit der Schluss ziehen, dass eine Kombination verschiedener Lernparadigmen, dem menschlichen Lernen ähnlich, durchaus das Potential aufweist, erhebliche Vorteile zu erzielen und es somit möglich ist, die Intuition des menschlichen Lernens zunächst auf naive Art und Weise auf maschineller Ebene abzubilden.

Im Rahmen dieser Untersuchungen gab es allerdings auch einige Resultate, welche gegen die Intuition strebten. So zum Beispiel die steigende Exploration irrelevanter (Zustand, Aktion)-Paare des Makro-Q-Learnings, um die Exploration des Labyrinths zu vereinheitlichen. Eine mögliche Verbesserung zum hier verwendeten Ansatz stellten Shie Mannor et al. in ihrer Publikation *„Dynamic Abstraction in Reinforcement Learning via Clustering"* vor. In diesem Ansatz werden Zustandsmengen derart zusammengefasst, sodass die resultierenden Cluster intuitiv betrachtet Räumen entsprechen und anschließend das Problemszenario in mehrere Teilprobleme (*das erreichen des nächsten Raumes*) aufgegliedert wird [16]. Anschließend wird zur Lösung des Problems nur noch die Information benötigt, welcher Raum zum Ziel führte und welcher nicht. Anhand dieses Ansatzes könnten zwei Probleme, welche im Rahmen dieser Untersuchungen aufgetreten sind, gelöst werden: zum einen liegt das Augenmerk nochmals verstärkt auf den Randzuständen dieser Cluster und alle Zustände im Inneren führen zum Rand, zum anderen könnte durch die Aufgliederung in n-Teilprobleme das Problem des *temporal credit assignments* gelöst werden, da die einzelnen Räume selbst als Zustände bewertet werden können und die Problemdimension somit reduziert wird. Weitere Ansätze zur Verbesserung der hier vorgestellten Resultate verfolgen zum einen das *Reward shaping*, bei welchem die Belohnung einer jeden Aktion im Verlauf der Lernphase angepasst wird und somit zum zielführende (Zustand, Aktion)-Paare eine bessere Bewertung über den Verlauf des Lernprozesses erhalten als andere [13], sowie zum anderen das *intrinsic motivated Reinforcement Learning* [25], [21]. Dieser Ansatz versucht die menschliche Motivation in das maschinelle Lernen einzubetten und könnte im Rahmen dieses Experiments in zweierlei Richtungen arbeiten: zum einen könnte zu Beginn eine erhöhte Motivation bestehen, um das Labyrinth zu explorieren, und zum anderen eine über den Lernverlauf des Agenten steigende Motivation, die erzielten Erfolge auszunutzen und somit eine stärkere Exploitation zu erzielen, um das Ziel schneller zu erreichen.

Literaturverzeichnis

[1] AIBO, Artificial Intelligence roBOt. http://www.electronicpets.org/sony-aibo-ers7~p14.html.

[2] Definition des Wortes somatosensorisch. http://flexikon.doccheck.com/Somatosensorisch.

[3] RoboCup, Internationale Wettbewerbe zum foerdern der Weiterentwicklung intelligenter Roboter. http://www.ais.uni-bonn.de/robocup.de/.

[4] Roboter suchen nach Erdbebenopfern. http://www.golem.de/1103/82068.html.

[5] Roboterfußball. http://www.ais.uni-bonn.de/robocup.de/soccer.html.

[6] Robots for humanity. http://www.willowgarage.com/blog/2011/07/13/robots-humanity.

[7] Sagen um Rabbi Judah Loew und seinen Golem aus Lehm. http://www.schwarzaufweiss.de/Prag/golem.htm.

[8] Ursula Brandstaetter. *Bildende Kunst und Musik im Dialog: aesthetische, zeichentheoretische und wahrnehmungspsychologische Ueberlegungen zu einem kunstspartenuebergreifenden Konzept aesthetischer Bildung*. Wissner-Verlag, 2004.

[9] Advait Jain Charles C. Kemp Chih-Hung King, Tiffany L. Chen. Towards an Assistive Robot that Autonomously Performs Bed Baths for Patient Hygiene.

[10] Kenji Doya. What are the computations of the cerebellum, the basal ganglia and the cerebral cortex? 1999. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.178.2123&rep=rep1&type=pdf.

[11] Walter Edelmann. *Lernpsychologie*. BeltzPVU, 2000.

[12] Gao et al. Cerebellum implicated in sensory acquisition and discrimination rather than motor control. 1996.

[13] Marek Grześ and Daniel Kudenko. Online learning of shaping rewards in reinforcement learning. 2010.

[14] R. B. Ivry. The representation of temporal information in perception and motor control. 1996.

[15] Bibliographische Institut Mannheim. Lexirom, 1999.

[16] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering.

[17] Amy McGovern and Richard S. Sutton. Macro-actions in reinforcement learning: An empirical analysis. 2005.

[18] Amy McGovern, Richard S. Sutton, and Andrew H. Fagg. Roles of macro-actions in accelerating reinforcement learning. 1997.

[19] RC Miall, DJ Weir, DM Wolpert, and JF Stein. Is the cerebellum a smith predictor? 1993. http://prism.bham.ac.uk/pdf_files/SmithPred_93.PDF.

[20] Rosemarie Mielke. *Psychologie des Lernens: Eine Einfuehrung*. Kohlhammer, 2001.

[21] Scott Niekum. Evolved Intrinsic Reward Functions for Reinforcement Learning.

[22] Michael G. Paulin. The role of the cerebellum in motor control and perception. 1993. http://www.otago.ac.nz/neurozoo/documents/Paulin%201993%20-%20role%20of%20the%20cerebellum.pdf.

[23] Guenter Schmitt. Skript zu Lernen und Verhaltensaenderung, 1999.

[24] W. Schultz. Predictive reward signal of dopamine neurons. 1998.

[25] Satinder Sing, Andrew G. Barto, and Nuttapong Chentanez. Intrinsically Motivated Reinforcement Learning.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. MIT Press, 1998.

[27] Richard S. Sutton, Doina Precub, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. 1999.

[28] P. Apicella und T. Ljungberg W. Schultz. Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. 1998.

[29] Christopher J. C. H. Watkins. Learning from delayed rewards. 1989.

[30] Christopher J. C. H. Watkins and Peter Dayan. Technical Note Q-Learning, 1992. `http://www.springerlink.com/content/t774414027552160/`.

[31] Uri Wolf, Mark J. Rapoport, and Tom A. Schweizer. Evaluating the affective component of the cerebellar cognitive affective syndrome. 2009. `http://neuro.psychiatryonline.org/article.aspx?articleID=103781`.

Abbildungsverzeichnis