Erweiterung einer Reinforcement-Learning Software um Q-Learing, Sarsa und verschiedene Explorationsstrategien. Entwicklung eines GridWorld-Editors

Tobias Bystricky

December 24, 2009

Contents

1 Augabenstellung

Im Rahmen einer Diplomarbeit [Tok06] wurde eine Software entwickelt, die den Lernalgorithmus Value Iteration simuliert und den Ablauf des Verfahren grafisch darstellt. Zielsetzung meiner Arbeit sollte die Erweiterung dieser Software um die Lernverfahren Q-Learning sowie Sarsa [SB98] sein. Bei den Lernverfahren sollte das Auswählen verschiedener Explorationsstrategien möglich sein. Als Policies sollte zwischen E-Greedy und Value Difference Based Exploration (VDBE) [Tok07] gewählt werden können. Die Speicherung der Q-Werte sollte zum Einen in einer Wertetabelle, zum Anderen in einem neuronalen Netz möglich sein. Die grafische Darstellung der Verfahren sollte über die bereits vorhandenen Methoden statt finden.
Im zweiten Teil der Arbeit sollte für das Forschungsprojekt ZAFH [ZAF] eine Software um eine Gridworld erweitert werden. Ziel der Arbeitsgruppe Ertel im ZAFH-Projekt ist die Entwicklung einer universell einsetzbaren Teachingbox [ESCT09], die es ermöglichen soll, dass zukünftige Serviceroboter ihr Verhalten erlernen können. Die Teachingbox ist eine Softwareumgebung in der Experimente mit verschiedenen Lernverfahren (Agenten) und unterschiedlichen Umgebungen(Environment) durchgeführt werden können. Agenten stellen in der Teachingbox unter Anderem den Lernalgorithmus dar. Neben Q-Learning können z.B. auch ValueIteration und SARSA ausgewählt werden. Zudem können verschiedene Explorationsverfahren wie z. B. Greedy, Softmax oder Egreedy gewählt werden. Jedes Experiment benötigt ein Umfeld(Environment). Das Enviroment gibt dem Agenten nach jedem Schritt ein Reward und den Folgezustand zurück. Enviroments können simulierte Probleme wie z.B. Pole Swing-Up oder Gridworld oder Hardwarelösungen wie z.B. der Laufroboter sein. Ziel dieser Arbeit soll es sein, ein Environment zu entwickeln das eine Gridworld darstellt. Dem Benutzer soll die Möglichkeit gegeben werde, die Gridworld zu editieren. Dabei sollen Rewards eingegeben werden können. Ausserdem soll die Möglichkeit gegeben werden, verschiedene Hindernisse(Wand oder Hinderniss) in der Gridworld zu platzieren. Besondere Zustände wie der absorbiernde Endzustand sollen eingefügt werden können. Eine ansprechende grafische Gestaltung ist gewün-

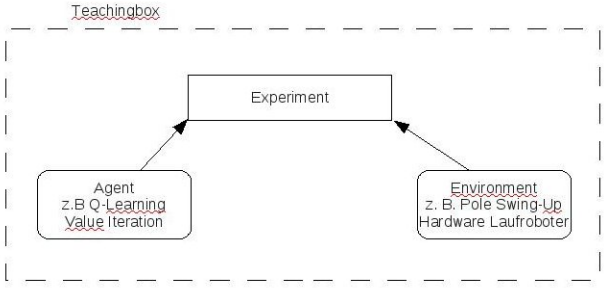


Figure 1: Umgebung teachingbox

scht.

2 Erweiterung Reinforcement-Learning Software

Um die Software strikt objektorientiert weiter zu entwickeln, wurde zunächste ein Model der Gridworld geschaffen. Das Model enthält zum Einen alle Daten der Gridworld, wie z.B. das Feedback oder die Q-Values. Zum Anderen wurden alle nötigen Funktionen zur Manipulation des Models auch direkt im Model implementiert. Um die beiden Lernverfahren (Q-Learning, Sarsa) parallel ausführen zu können wurde für jedes Lernverfahren ein eigener Thread erstellt. Ändert der Benutzer das Lernverfahren, wird das abgewählt Lernverfahren "eingefroren" und kann später wieder fortgesetzt werden. Das Lernverfahren kann also ausgetauscht werden.

2.1 Lernverfahren

Die folgenden Lernverfahren wurden in das Model der Gridworld integriert. Die beiden Lernverfahren Q-Learning und Sarsa werden beide sehr gut und detailiert im Buch "Reinforcement Learning: An Introduction" [SB98] Kapitel 6.4 und Kapitel 6.5 beschrieben. Daher wird in dieser Arbeit nur noch auf die für die Arbeit relevanten Details eingegangen.

Figure 2: UML-Darstellung des Models

2.1.1 Q-Learning

Die Update-Regel von QLearning:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \qquad (1)$$

Bei dieser Regel wird für das Update immer der aktuelle Reward (r_{t+1}), der aktuelle Q-Value ($Q(s_t, a_t)$) und die Aktion mit dem höchsten Q-Value des nächsten Zustandes verwendet. (Off-Policy Verfahren)
Die Implementierung des QLearning-Algorithmus sieht wie folgt aus.

```
1 //Holen der Richtung des naechsten Schritts
2 int dir = instanz->getDirection(instanz->actState);
3 //Holen des dazugehoerigen Feedbacks
4 int dir_feedback = instanz->getActFeedback(dir);
```

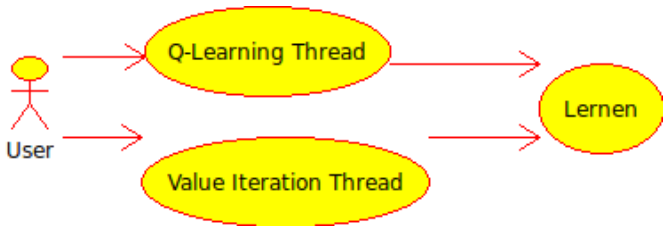


Figure 3: Threads

```
//Holen des dazugehoerigen Q-Wertes
double old_qValue = instanz->getActQValue(dir);
//Naechsten Zustand holen
QState *nextState = new QState();
instanz->getNextState(dir, nextState);
//Holen des groe ten QWerts aus dem n chsten Zustand
double maxNextQValue = instanz->getMaxQValue(nextState);
//Neuen QWert berechnen
double qValue = old_qValue + instanz->alpha * (dir_feedback +
              instanz->gamma * maxNextQValue - old_qValue);
//Neuen QWert setzen
instanz->setQValue(instanz->actState, dir, qValue);
//In naechsten Zustand springen
instanz->setActState(nextState->row, nextState->col);
delete(nextState);
```

2.1.2 SARSA

Die Update-Regel von SARSA:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \tag{2}$$

Bei dieser Regel wird für das Update immer der aktuelle Reward (r_{t+1}), der aktuelle Q-Value ($Q(s_t, a_t)$) und der Q-Value des nächsten Zustandes welches der Policy folgt, verwendet. Dies bedeutet, dass nicht immer die Aktion mit dem höchsten Q-Value verwendet wird, sondern auch, abhängig von ε, eine zufällige Aktion gewählt werden kann. (On-Policy Verfahren)
In C++ wurde die Regel wie folgt umgesetzt.

```
int dir;
//Wenn noch keine Richtung definiert
if(instanz->nextDirection==-1){
    dir = instanz->getDirection(instanz->actState);
}else{
```

```
6       dir = instanz->nextDirection;
7   }
8   //Feedback f r den naechsten Step holen
9   int dir_feedback = instanz->getFeedback(instanz->actState->row,
10                        instanz->actState->col, dir);
11  //aktuellen QWert holen
12  double old_qValue = instanz->getActQValue(dir);
13  //Naechsten Zustand holen
14  QState* nextState = new QState();
15  instanz->getNextState(dir, nextState);
16  //Richtung im naechsten Zustand bestimmen
17  int newDir = instanz->getDirection(nextState);
18  instanz->nextDirection = newDir;
19  //QWert des naechsten Zustandes bestimmen
20  double nextQValue = instanz->getQValue(nextState, newDir);
21  //Eigenen QWert berechnen
22  double qValue = old_qValue + instanz->alpha * (dir_feedback +
23       instanz->gamma * nextQValue - old_qValue);
24  //Eigenen QWert setzen
25  instanz->setQValue(instanz->actState, dir, qValue);
26  //In naechsten Zustand springen
27  instanz->setActState(nextState->row, nextState->col);
28  delete(nextState);
```

2.2 Explorationsverfahren

Für jedes Lernverfahren sollte die Auswahl verschiedener Explorationsverfahren möglich sein. Die Explorationsverfahren bestimmen die Auswahl der nächsten Aktion. Im Folgenden werden die Verfahren und ihre Implementierung kurz vorgestellt.

2.2.1 E-Greedy

Bei diesem Verfahren wird ein konstantes und globales Epsilon verwendet ($0 \leq \varepsilon \leq 1$). Das Epsilon bestimmt den Anteil an zufällig ausgewählten Aktionen. Für die Implementierung musste dazu nur die Funktion, welche die nächste Richtung bestimmt, abgeändert werden.

```
1   if (rand()%100 < instanz->epsilon) {
2       while (instanz->testBorders(greedyAction, state)==false) {
3           greedyAction = rand() % 4;
4       }
5       return greedyAction;
6   }else{
```

```
7       return normalAction
8   }
```

2.2.2 Value Difference Based Exploration

Dieses Verfahren wurde von Michel Tokic in einer Projektarbeit entwickelt [Tok07]. Bei diesen Verfahren gibt es kein globales ε sondern für jeden Zustand ein lokales ε. Die lokalen ε werden mit 0 initalisiert. Nach jeder Aktion wird das ε der Aktion aktualisiert. Das Update hat folgende Form:

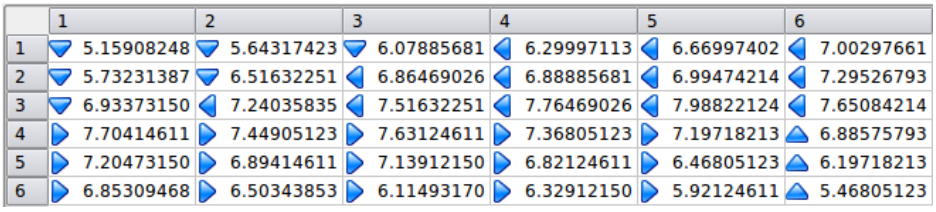
$$\epsilon = \sqrt{\frac{|Q_{t+1} - Q_t|}{|Q_{t+1} + Q_t|}} \tag{3}$$

Diese Update wurde wie folgt implementiert.

```
1   double part1 = (qValue-old_qValue);
2   if(part1 < 0){
3       part1 *=-1;
4   }
5   double part2 = (qValue+old_qValue);
6   if(part2 < 0){
7       part2 *=-1;
8   }
9   if(part2 == 0){
10      instanz->memory[actState->row][actState->col].epsilon = 0;
11  }else{
12      instanz->memory[actState->row][actState->col].epsilon =
13                              sqrt(part1/part2);
14  }
```

2.3 Anpassungen an der GUI

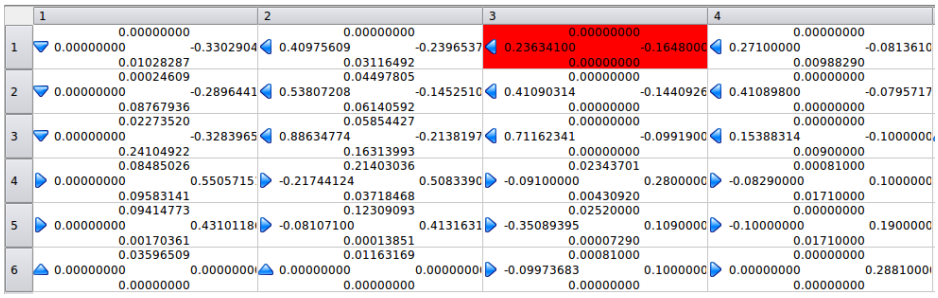
Neben der Implementierung verschiedener Lernverfahren, musste auch die GUI an die neuen Verfahren angepasst werden. Bisher wurde für die Anzeige der Werte für Value Iteration eine einfache Tabelle verwendet. Für das Q-



	1	2	3	4	5	6
1	5.15908248	5.64317423	6.07885681	6.29997113	6.66997402	7.00297661
2	5.73231387	6.51632251	6.86469026	6.88885681	6.99474214	7.29526793
3	6.93373150	7.24035835	7.51632251	7.76469026	7.98822124	7.65084214
4	7.70414611	7.44905123	7.63124611	7.36805123	7.19718213	6.88575793
5	7.20473150	6.89414611	7.13912150	6.82124611	6.46805123	6.19718213
6	6.85309468	6.50343853	6.11493170	6.32912150	5.92124611	5.46805123

Figure 4: Value Iteration Value Table

Learning mit einer E-Greedy Policy wurde die Tablelle auf vier Aktionen pro Zelle erweitert. Ausserdem wurde für das bessere Verständnis des Lernalgorithmus die aktive Zelle farbig markiert. Rot steht für eine Aktion on-policy. Gelb ist eine zufällige Bewegung. Um die Konfiguration für den Benutzer



	1	2	3	4
1	0.00000000 0.00000000 -0.3302904 0.01028287	0.00000000 0.40975609 -0.2396537 0.03116492	0.00000000 0.23634100 -0.164800 0.00000000	0.00000000 0.27100000 -0.0813610 0.00988290
2	0.00024609 0.00000000 -0.2896441 0.08767936	0.04497805 0.53807208 -0.1452510 0.06140592	0.00000000 0.41090314 -0.1440926 0.00000000	0.00000000 0.41089800 -0.0795717 0.00000000
3	0.02273520 0.00000000 -0.3283965 0.24104922	0.05854427 0.88634774 -0.2138197 0.16313993	0.00000000 0.71162341 -0.0991900 0.00000000	0.00000000 0.15388314 -0.1000000 0.00900000
4	0.08485026 0.00000000 0.5505715 0.09583141	0.21403036 -0.21744124 0.5083390 0.03718468	0.02343701 -0.09100000 0.2800000 0.00430920	0.00081000 -0.08290000 0.1000000 0.01710000
5	0.09414773 0.00000000 0.4310118 0.00170361	0.12309093 -0.08107100 0.4131631 0.00013851	0.02520000 -0.35089395 0.1090000 0.00007290	0.00000000 -0.10000000 0.1900000 0.01710000
6	0.03596509 0.00000000 0.0000000 0.00000000	0.01163169 0.00000000 0.0000000 0.00000000	0.00081000 -0.09973683 0.1000000 0.00000000	0.00000000 0.00000000 0.2881000 0.00000000

Figure 5: Q-Learning Egreedy Value Table

so einfach wie möglich zu gestalten wurde eine intuitiv bedienbare Oberfläche gestaltet. Die Werte für γ, α und die Explorationsrate ε können durch den Benutzer numerisch angegeben werden. Die Auswahl des Lernverfahrens bzw. der Explorationsmethodik erfolgt durch Checkboxen. Als Startzustand kann ein Standartzustand (0/0) oder ein zufällig gewählter Zustand verwendet werden. Als letzte Möglichkeit kann der Benutzer entscheiden, ob die Q-Values in der Memory Table gespeichert oder durch ein neuronalen Netz approximiert werden. Sollen die Werte durch ein neuronalen Netz repräsentiert

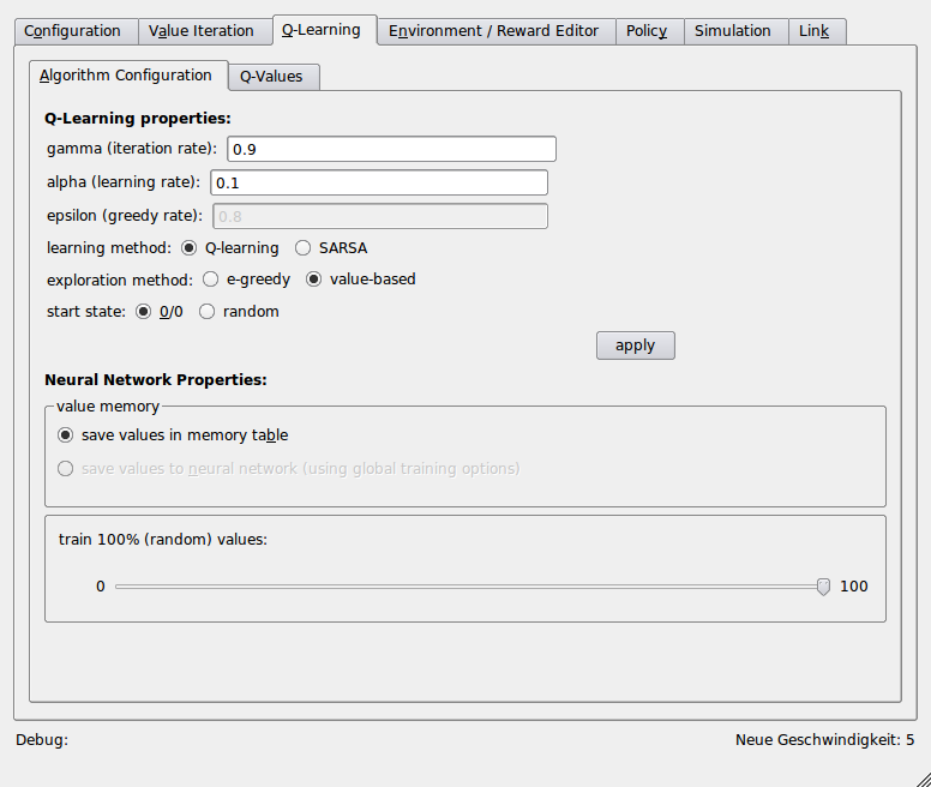


Figure 6: Der Q-Learning Konfigurations Dialog

werden, so ist eine Konfiguration des Netzes auf einem extra Tab möglich. Die Geschwindigkeit der Lernprozesses kann über einen Regler eingestellt werden.

3 Entwicklung eines GridWorld-Editors

Im zweiten Teil der Projektarbeit wurde ein neues Environment für die Teachingbox [ESCT09] geschaffen. Die Teachingbox besteht aus verschiedenen Modulen die beliebig zusammen gesetzt werden können. Um ein Experiment zu erstellen, benötigt man unter anderen ein Environment. Das Environment kann eine reale Hardware oder eine Softwaresimulation sein. Ziel dieser Arbeit soll die Entwicklung eines Environment sein, welches eine Gridworld simuliert. Um die Rewards sowohl manuell Eingeben zu können als auch aus einer Datei zu laden, wurde eine grafische Oberfläche entwickelt. Zur Daten-

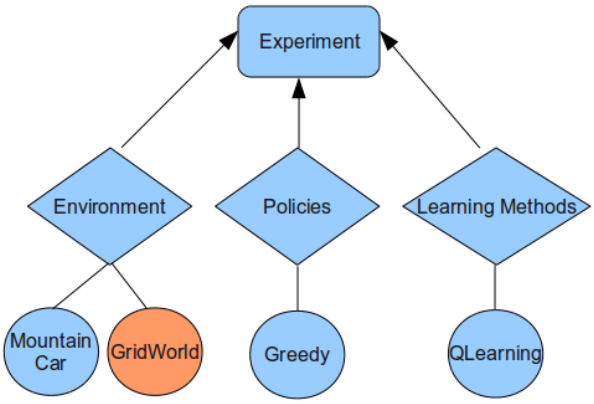


Figure 7: Teachingbox Umfeld

haltung wurde ein GridModel geschaffen, welches alle Daten der Gridworld enthält. Die GUI ist im Prinzip nur die grafische Umsetzung des Models. Der typische Verlauf einer Lernfolge wird in der folgenden Zeichnung dargestellt.

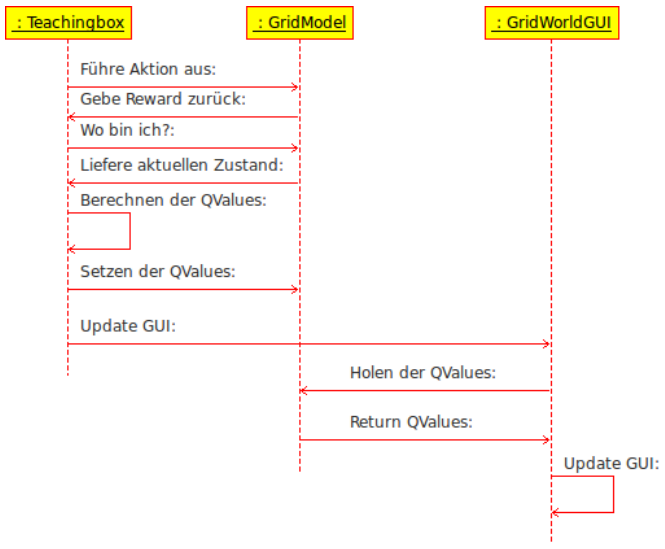


Figure 8: Sequenzdiagramm Ablauf GridEditor

3.1 Das GridModel

Zur Datenhaltung wurde das GridModel geschaffen. Dies enthält neben den globalen Daten eine Tabelle in der alle Rewards und Q-Values gehalten werden. Globale Daten sind:

- SleepingTime: Zeit die das Model wartet bevor ein Reward zurück gegeben wird.
- ActCol: Spalte des aktuellen Zustandes.
- ActRow: Zeile des aktuellen Zustandes.
- Size: Größe der GridWorld.

Die Tabelle enthält die Zustände (GridCell) der GridWorld in einem zweidimensionalen Array. Jeder Zustand des Models enthält zustandsglobale Daten:

- col: Spalte des Zustandes.
- row: Zeile des Zustandes.
- isTerminalState: Definiert den Zustand als Endzustand.
- isStartState: Definiert den Zustand als Startzustand.
- wall[]: Für jede Richtung des Zustandes kann ein Hinderniss gesetzt werden. Dieses verhindert, dass eine Aktion in diese Richtung ausgeführt werden kann. Soll nun trotzdem eine Aktion in Richtung der Wand ausgeführt werden, wird der Reward zurück gegeben. Allerdings wird der aktuelle Zustand nicht verändert.

Erwähnenswert ist noch die Funktion double doAction(Action a) im GridModel. Diese Funktion führt einen Schritt in der GridWorld aus. Zuerst wartet die Funktion die zuvor global im GridModel definierte Sleeptime. Dann wird überprüft ob die Aktion möglich ist. Sind keine Wände in der gewünschten Richtung wird der Reward der Aktion zurück gegeben.
Im Sinne der Objektorientierung wurden Get()- und Set()-Methoden für alle

globalen Daten erzeugt.
Um den Zugriff auf das Model so einfach wie möglich zu gestalten wurde das Singleton-Entwurfsmuster [SIN] verwendet. Dabei wird nur eine einzige Instanz des Models erzeugt. Der Zugriff auf die Instanz erfolgt über eine statische Methode. Mit Hilfe der setSize()-Methode wird das Model initial-

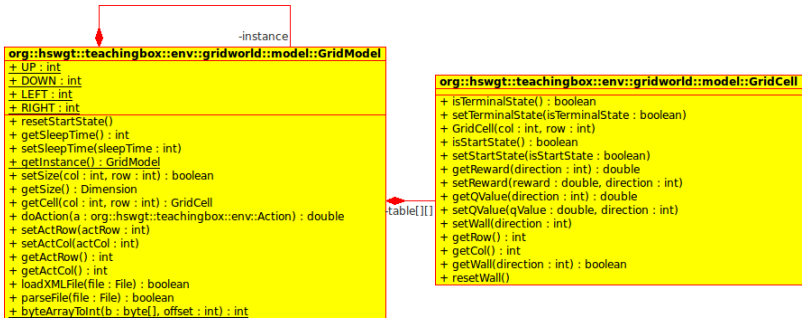


Figure 9: UML GridModel

isiert. Dabei werden alle Werte mit 0 initialisiert. Alle Hindernisse in den einzelnen Zuständen werden zurück gesetzt. Nach dem Initialisieren kann mit dem Model gearbeitet werden. Dazu werden verschiedene Set()-Methoden zur Verfügung gestellt. Um auch das Laden Verschiedene Dateitypen zu ermöglichen wurden auch Methoden zum Einlesen von Dateien erstellt. Es werden zwei verschiedene Dateitypen unterstützt.

3.1.1 .feedback

Dieses Dateiformat entstand in der Diplomarbeit von Michel Tokic [Tok06]. In dieser Datei werden die Rewards binär gespeichert. In der folgenden Abbildung wird der Aufbau der Datei dargestellt. Nach dem Header der aus

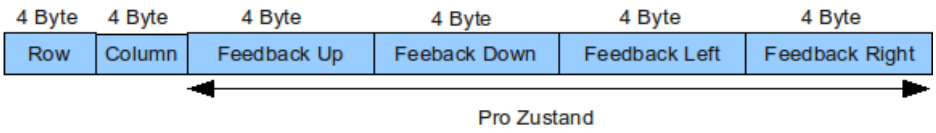


Figure 10: Format der Feedback-Datei

zwei mal 4 Byte, folgen die Zustände mit den Rewards für jede Richtung.

Das erste Byte im Header steht für die Anzahl der Zeilen. Das zweite Byte steht für die Anzahl der Spalten. Danach folgen zeilenweise alle Zustände mit den zugehörigen Feedbacks.
Der Nachteil dieses Formats ist, dass keine Hindernisse gespeichert werden und dass ein Editieren der Datei sehr schwierig ist.

3.1.2 .gridworld

Speziell für den Gridworld-Editor wurde ein neues Dateiformat auf XML-Basis erstellt. Das Schema für das Dateiformat hat folgenden Aufbau.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
       elementFormDefault="qualified">
3    <xs:element name="GridWorld">
4      <xs:complexType>
5        <xs:sequence>
6          <xs:element maxOccurs="unbounded" ref="State"/>
7        </xs:sequence>
8        <xs:attribute name="cols" use="required" type="xs:integer"/>
9        <xs:attribute name="rows" use="required" type="xs:integer"/>
10     </xs:complexType>
11   </xs:element>
12   <xs:element name="State">
13     <xs:complexType>
14       <xs:sequence>
15         <xs:element ref="Up"/>
16         <xs:element ref="Down"/>
17         <xs:element ref="Left"/>
18         <xs:element ref="Right"/>
19       </xs:sequence>
20       <xs:attribute name="col" use="required" type="xs:integer"/>
21       <xs:attribute name="row" use="required" type="xs:integer"/>
22     </xs:complexType>
23   </xs:element>
24   <xs:element name="Up">
25     <xs:complexType>
26       <xs:sequence>
27         <xs:element ref="reward"/>
28         <xs:element ref="wall"/>
29       </xs:sequence>
30     </xs:complexType>
31   </xs:element>
32   <xs:element name="Down">
33     <xs:complexType>
34       <xs:sequence>
35         <xs:element ref="reward"/>
```

```
36          <xs:element ref="wall"/>
37        </xs:sequence>
38      </xs:complexType>
39    </xs:element>
40    <xs:element name="Left">
41      <xs:complexType>
42        <xs:sequence>
43          <xs:element ref="reward"/>
44          <xs:element ref="wall"/>
45        </xs:sequence>
46      </xs:complexType>
47    </xs:element>
48    <xs:element name="Right">
49      <xs:complexType>
50        <xs:sequence>
51          <xs:element ref="reward"/>
52          <xs:element ref="wall"/>
53        </xs:sequence>
54      </xs:complexType>
55    </xs:element>
56    <xs:element name="reward" type="xs:decimal"/>
57    <xs:element name="wall" type="xs:integer"/>
58  </xs:schema>
```

Der Vorteil dieses Dateiformats ist die leichte Editierbarkeit sowie das Einhalten von Standards. Damit wird eine maximale Kompatibilität gewährleistet.

3.1.3 Arbeiten mit dem Model

Der Vorteil des GridModels ist, dass ein Arbeiten mit dem Models ohne GUI möglich ist.

Ohne GUI Im Folgenden wird kurz dargestellt wie man das Model ohne GUI mit dem Enviroment Interface der Teachingbox arbeitet.

```
1  package org.hswgt.teachingbox.env;
2
3  public class QonlyModel implements Environment{
4
5      public static final Action LEFT = new Action(new double[] { -1, 0 });
6      public static final Action RIGHT = new Action(new double[] { +1, 0 });
7      public static final Action UP = new Action(new double[] { 0, +1 });
8      public static final Action DOWN = new Action(new double[] { 0, -1 });
9
10     public static final ActionSet ACTION_SET = new ActionSet(new Action[] {
11             LEFT, RIGHT, UP, DOWN });
12
```

```
13      public static final ActionFilter FILTER = new ActionFilter() {
14          private static final long serialVersionUID = -5057734785625735742L;
15
16          public boolean isPermitted(final State s, final Action a){
17              if( s.get(0) <= 0 && a.equals(LEFT) ) return false;
18              if( s.get(0) >= GridModel.getInstance().getSize().getWidth() &&
                    a.equals(RIGHT) ) return false;
19              if( s.get(1) <= 0 && a.equals(UP) ) return false;
20              if( s.get(1) >= GridModel.getInstance().getSize().getHeight() &&
                     a.equals(DOWN) ) return false;
21              return true;
22          }
23      };
24
25      static {
26          ACTION_SET.setFilter( FILTER );
27      }
28
29      public QonlyModel() {
30          GridModel.getInstance().loadXMLFile(new
                File("/home/tobiby/Desktop/cliffwalk-4x12.gridworld"));
31          GridModel.getInstance().setSleepTime(0);
32      }
33
34      public double doAction(Action a) {
35          double value = GridModel.getInstance().doAction(a);
36          return value;
37      }
38
39      public State getState() {
40          return new State(new double[] {
41                  new
                        Double(GridModel.getInstance().getActCol()).doubleValue(),
42                  new
                        Double(GridModel.getInstance().getActRow()).doubleValue()
                        }).copy();
43      }
44
45      public void init(State s) {
46          GridModel.getInstance().setActCol(new Double(s.get(0)).intValue());
47          GridModel.getInstance().setActRow(new Double(s.get(1)).intValue());
48      }
49
50      public void initRandom() {
51          Random ran = new Random();
52          if (GridModel.getInstance().getSize().width != 0 &&
                GridModel.getInstance().getSize().height != 0) {
53              GridModel.getInstance().setActCol(ran.nextInt(GridModel.getInstance().getSize().width
                     - 1));
```

```
54              GridModel.getInstance().setActRow(ran.nextInt(GridModel.getInstance().getSize().heigh
                    - 1));
55          }
56
57      }
58
59      public boolean isTerminalState() {
60          return false;
61      }
62
63      public static void Display(State s, QFunction Q) throws Exception
64      {
65          GridModel.getInstance().getCell(new Double(s.get(0)).intValue(), new
                Double(s.get(1)).intValue()).setQValue(Q.getValue(s, UP),
               GridModel.UP);
66          GridModel.getInstance().getCell(new Double(s.get(0)).intValue(), new
                Double(s.get(1)).intValue()).setQValue(Q.getValue(s, DOWN),
               GridModel.DOWN);
67          GridModel.getInstance().getCell(new Double(s.get(0)).intValue(), new
                Double(s.get(1)).intValue()).setQValue(Q.getValue(s, LEFT),
               GridModel.LEFT);
68          GridModel.getInstance().getCell(new Double(s.get(0)).intValue(), new
                Double(s.get(1)).intValue()).setQValue(Q.getValue(s, RIGHT),
               GridModel.RIGHT);
69      }
70  }
```

Mit GUI Um das Model grafisch darzustellen ist es notwendig, die GUI im Konstruktor zu initalisieren. Dafür fällt der Aufruf der setSize-Funktion weg.

```
1  public GridworldEnvironment() {
2     gridWorld = new GridWorldGUI("src/gridworld.xml");
3     //GridModel.getInstance().loadXMLFile("test.gridworld");
4  }
```

Zusätzlich muss die GUI informiert werden, wenn das Model aktualisiert wurde. Dies geschieht am Besten in der getState-Funktion.

```
1  public State getState() {
2    gridWorld.refreshTable();
3    return new State(new double[] {
4      new Double(GridModel.getInstance().getActCol()).doubleValue(),
5      new Double(GridModel.getInstance().getActRow()).doubleValue() }).copy();
6  }
```

3.2 Die GUI

Zur Darstellung der Gridworld wird ein JTableverwendet. Der JTable verwendet jedoch einen eigenes Model welches nur eine direkte Schnittstelle zum oben beschriebenen GridModel darstellt. Zur besseren Darstellung wurde ein eigener Renderer für die Tabellenzellen erstellt. Dieser ist zum Einen für die darstellung der QValues sowie für die farbliche Darstellung des Hintergrundes verantwortlich. In der folgenden Skizze sind alle Elemente aufgezeigt die zum darstellen der Tabelle benötigt werden.

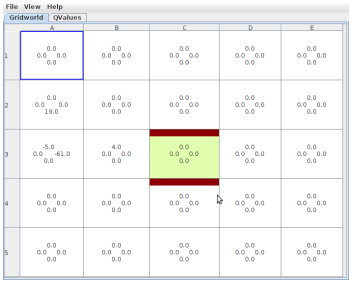


Figure 11: Gridworld-Editor

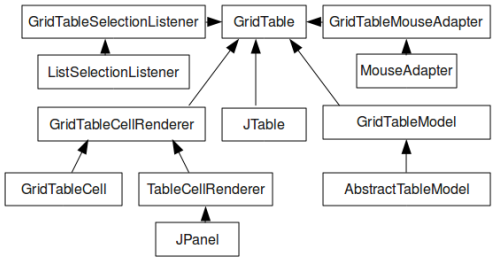


Figure 12: Aufbau der GridTable

Zum Setzen der Hindernisse wurd eine Menü entwickelt, das über die rechte Maustaste aktiviert. Somit kann der Benutzer sehr schnell Hindernisse setzen.
Neben der Tabelle welche die Rewards anzeigt, wurde diverse Toolbars entwickelt.
Alle Toolbars können beliebig in der Applikation plaziert werden.

3.2.1 SpeedToolbar

Dient zum Einstellen der Pausen zwischen der Abfrage von zwei Werten. Dadurch kann eine Hardware simuliert werden. Zusätzlich lässt sich das Lernverfahren besser nachvollziehen.

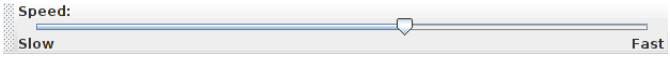


Figure 13: Speedtoolbar

3.2.2 RewardToolbar

Zum leichten Editieren der Rewards wurde ein eigener Toolbar geschaffen. Nach Selektieren der gewünschten Zustände in der Tabelle können über diesen Toolbar die Rewards editiert werden.

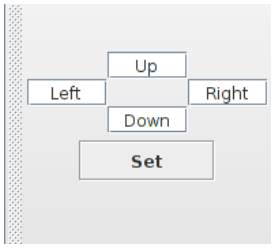


Figure 14: Rewardtoolbar

3.3 Nachforderungen

Trotz Absprachen gab es in dieser Arbeit einige Nachforderungen.

3.3.1 Anzeigen der QValues

Nachdem der eigentliche Editor entwickelt war, wurde der Wunsch geäussert die QValues aus der Teachingbox anzeigen zu können. Auch wenn das Berechnen der QValues Aufgabe der Teachingbox ist, sollten die QValues von dem externen Gridworld-Editor angezeigt werden.
Das Environment-Interface der Teachingingbox stellt eine Funktion zum Anzeigen der QValues zur Verfügung. Diese Funktion wurde entfremdet um die QValues in das GridModel zu laden. Das GridModel wird nun verwendet um die QValues anzuzeigen.
In der GUI wurde eine neue Tabelle zur Darstellung der QValues eingefügt. Diese entspricht vom Aufbau(Model, Renderer) der Rewardtabelle ist aber nicht editierbar.

3.3.2 Anbinden des Laufroboters

Für die Anbindung des Laufroboters aus dem ZAFH-Projekt mussten das Model und GUI noch ein wenig modifiziert werden. Auf das Anpassen

der Spalten- und Zeilengröße musste aufgrund von Performanceproblemen verzichtet werden.

4 Zusammenfassung

Aus einer überschaubaren Aufgabe wurde im Laufe der Arbeit ein großes Paket. Der Aufwand bestand jeweils darin in einem bestehenden System neue Komponenten einzubringen.
In der Simulationssoftware des Laufroboters war dies sehr schwierig, da das System in kurzer Zeit entwickelt wurde und der Quellcode dem entsprechend unübersichtlich war. Bis zuletzt gab es Probleme mit der Synchronisierung der verschiedenen Lernverfahren.
Die Entwicklung des Gridworld-Editor gestaltete sich einfacher, da ausser einem Interface keine strengen Vorgaben gegeben waren. Es wurde also eine Software geschaffen, die leicht Erweiterbar und leicht zu Bedienen ist. Dies hat die Anbindung des Laufroboters an den Gridworld-Editor gezeigt.

5 FAQ

5.1 Wie kann ich eine vorgefertigte GridWorld direkt beim Start der GUI laden?

Dazu gibt es in der Konfigurationsdatei gridworld.xml. In dieser Datei gibt es das Element <loadFilePath>. Dort kann der absolute Pfad der zu ladenen Datei eingtragen werden.
Beispiel:

```
1  <loadFilePath>/home/tobiby/Desktop/cliffwalk-4x12.gridworld</loadFilePath>
```

Es ist zu beachten, dass nicht die Funktion des GridModels loadXMLFile(File file); im Zusammenspiel mit der GUI verwendet werden darf.

5.2 Kann man, um die Performance zu verbessern, nur auf dem Model arbeiten?

Dies ist grundsätzlich möglich. Ein Environment welches nur auf dem Model arbeitet ist im Kapitel 3.1.3 beschrieben und im Subversion im Environment "QonlyModel.java" zu finden.

References

[ESCT09] Wolfgang Ertel, Markus Schneider, Richard Cubek, and Michel Tokic. The teaching-box: A universal robot learning framework. In *In Proceedings of the 14th International Conference on Advanced Robotics (ICAR 2009)*, pages 1–6, 2009. http://www.servicerobotik.hs-weingarten.de/teachingbox. 1, 3

[SB98] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. 1, 2.1

[SIN] Singleton Entwurfsmuster. Website. http://de.wikipedia.org/wiki/Singleton_(Entwurfsmuster) [Online; accessed 17-November-2009]. 3.1

[Tok06] Michel Tokic. Entwicklung eines lernenden laufroboters. Diplomarbeit, Hochschule Ravensburg-Weingarten, Doggenriedstrasse, 88250 Weingarten, GERMANY, August 2006. 1, 3.1.1

[Tok07] Michel Tokic. Optimierung des explorationsverhaltens eines lernenden laufroboters. Scientific project, University of Applied Sciences Ravensburg-Weingarten, Weingarten, GERMANY, September 2007. 1, 2.2.2

[ZAF] ZAFH - Autonome Mobile Serviceroboter. Website. http://zafh.hs-weingarten.de [Online; accessed 26-October-2009]. 1