Praktikum: Selbstlernende Systeme

Organisatorisches/Reinforcement Learning

16.10.2019

Universität Augsburg
Institut für Informatik
Lehrstuhl für Organic Computing

Gliederung

ORGANIC COMPUTING

1. Organisatorisches
2. StarCraft II
3. Reinforcement Learning
4. Q-Learning
5. Aufgaben
6. Quellen

Organisatorisches

Termine

ORGANIC COMPUTING

- jeden Mittwoch 15:45 - 17:15
- Inhalt:
  - theoretische Grundlagen
  - Beantwortung von Fragen / Bearbeitung der Aufgaben

Fokus:

selbstständiges Arbeiten!!!

Termine

16.10.2019	Kick-Off & Stoff
23.10.2019	freies Arbeiten
30.10.2019	Stoff
06.11.2019	freies Arbeiten
13.11.2019	freies Arbeiten
20.11.2019	Stoff
27.11.2019	freies Arbeiten
04.12.2019	freies Arbeiten
11.12.2019	fällt aus
18.12.2019	Stoff
23.12 - 6.1	Weihnachtsferien
08.01.2020	Stoff - (Gruppenarbeit)
15.01.2020	freies Arbeiten
22.01.2020	freies Arbeiten
29.01.2020	freies Arbeiten
05.02.2020	freies Arbeiten

Bearbeitung der Aufgaben

ORGANIC COMPUTING

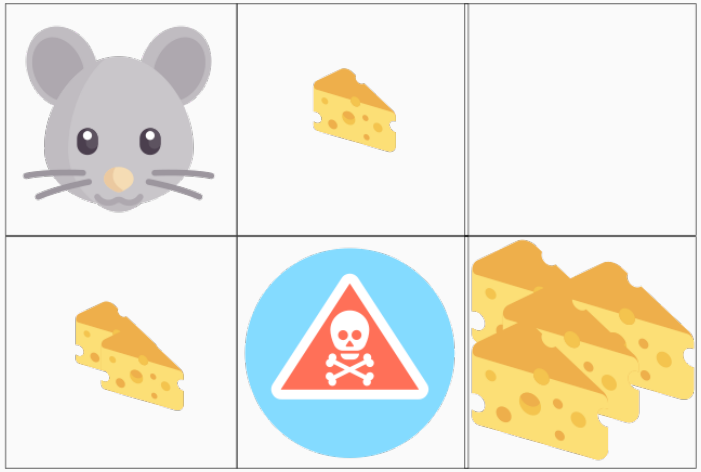
- Bearbeitung der Aufgaben alleine
- Letzte Aufgabe wird in 3er Gruppe bearbeitet
- Abschreiben –> 0 Punkte für das ganze Blatt für alle Beteiligten

WICHTIG: Erklären/Besprechen ist okay.
Abschreiben nicht.

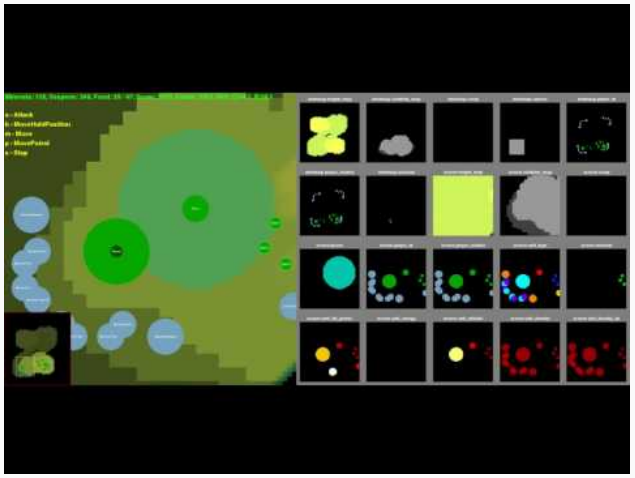
Thema

ORGANIC COMPUTING

- Reinforcement Learning



- StarCraft II API (Blizzard & Google)



Abgaben

ORGANIC COMPUTING

- regelmäßige Abgaben
- Email an:
  wenzel.pilar-von-pilchau@informatik.uni-augsburg.de
- gezippter Ordner: *<Gruppenname/RZ><#Abgabe>.zip*
  Inhalt:
  - Python-Projekt - Name: *<Gruppenname/RZ>*
  - Ordner - Name: *Schriftlich* - beinhaltet Evaluationen und schriftliche Antworten auf Fragen
  - Pdf-File - Name: *Protokoll-<Gruppenname><#Abgabe>.pdf* - beinhaltet ein Protokoll in dem notiert ist, welches Teammitglied welchen Teil der Lösung beigetragen hat

klare Empfehlung: **Linux**

- Visualisierung kann unter Windows/Mac nicht ausgeschaltet werden
- Teilweise einfachere Installation von Paketen unter Linux
- Spätestens bei der Gruppenaufgabe ist Linux nötig (1 Instanz)

Abgaben

ORGANIC COMPUTING

Anmerkungen zur Codequalität:

- Es ist nicht verboten Objekt-orientiert zu programmieren
- Im Projekt Ordner verwenden um Übersichtlichkeit zu gewinnen (Bsp. Aufgabe1, Learner, oä.)
- wichtige Stellen kommentieren
- sprechende Variablennamen
- Codequalität fließt in die Bewertung ein

Wichtig: Debugger benutzen

StarCraft II

PySC2

ORGANIC COMPUTING

- DeepMind's Python Framework
- Reinforcement Learning Environment
  - bekomme **Observation**
  - sende **Action**
- github: www.github.com/deepmind/pysc2

Deepmind

ORGANIC COMPUTING

- Wissen beschränkt auf das was menschlicher Spieler auch weiß
- Keine Chance gegen leichteste KI
- Für bekannte Methoden zu komplex

Stand 2019:
AlphaStar gewinnt 10:1 gegen professionelle StarCraft 2 Spieler

Minigames

ORGANIC COMPUTING

- isolierte Betrachtung von einzelnen Elementen des Spiels
- fokussierte Szenarios auf kleinen Karten
- Testen/Lernen von Aktionen und/oder Spielmechaniken

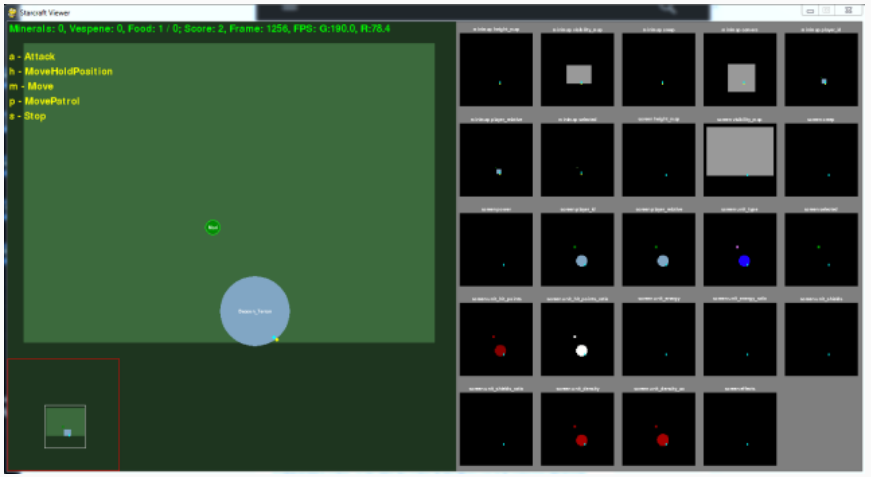
Minigames

- MoveToBeacon
- CollectMineralShards
- FindAndDefeatZerglings
- DefeatRoaches
- DefeatZerglingsAndBanelings
- CollectMineralsAndGas
- BuildMarines

Minigames im Praktikum

ORGANIC COMPUTING

Anwendung verschiedener Reinforcement Learning Algorithmen auf der Minimap **MoveToBeacon**



Units

ORGANIC COMPUTING

Marine:

```
1  def get_marine(obs):
2      marine = next(unit for unit in obs.observation.feature_units
3                    if unit.alliance == features.PlayerRelative.SELF)
4      return marine
```

Beacon:

```
1  def get_beacon(obs):
2      beacon = next(unit for unit in obs.observation.feature_units
3                    if unit.alliance == features.PlayerRelative.NEUTRAL)
4      return beacon
```

Position:

```
1  def getCoord(unit):
2      x = unit.x
3      y = unit.y
4      return x, y
```

Actions

ORGANIC COMPUTING

Keine Aktion:

```
1 return actions.FUNCTIONS.no_op()
```

Select Unit:

```
1 return actions.FUNCTIONS.select_army("select")
```

Move Selected Units:

```
1 return actions.FUNCTIONS.Move_screen("now", (x, y))
```

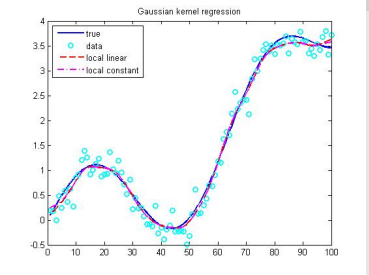
Reinforcement Learning

Machine Learning

ORGANIC COMPUTING

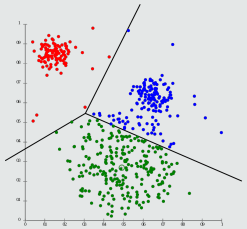
Supervised Learning

→ Aufgabengetrieben
(Regression / Klassifikation)



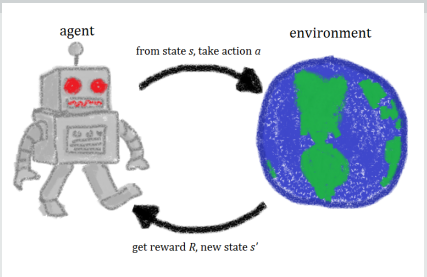
Unsupervised Learning

→ Datengetrieben
(Clustering)



Reinforcement Learning

→ reagiert auf Umgebung



Definition

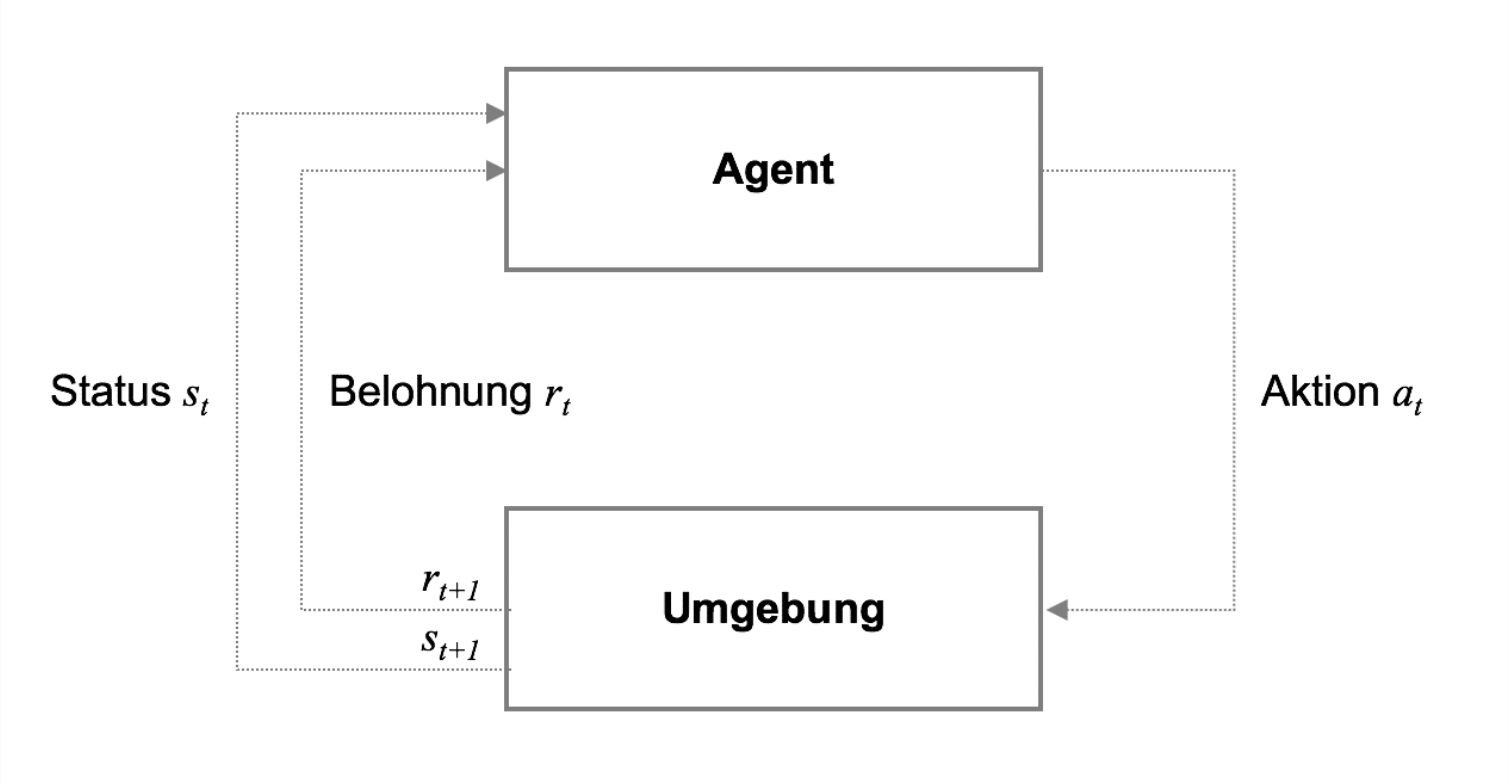
ORGANIC COMPUTING

Reinforcement Learning

”Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. These two characteristics—trial-and-error search and delayed reward—are the two most important distinguishing features of reinforcement learning.”

Richard S. Sutton and Andrew G. Barto

Komponenten



1. Agent
2. Environment
3. State
4. Action
5. Reward

Exploration-Exploitation Dilemma

ORGANIC COMPUTING

Ziel

Über *n* Iterationen den Zusammenhang zwischen Aktion und erwartetem Reward erkennen und den Reward maximieren

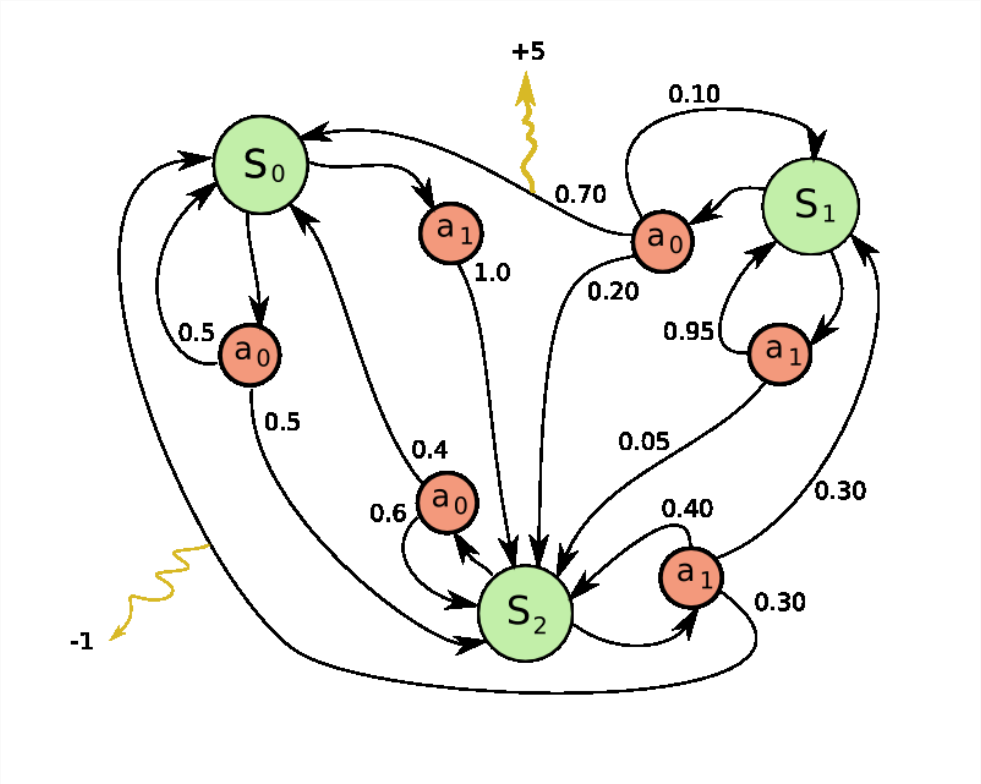
Problem: Bereits bekannte Strategie weiterverfolgen oder neue unbekannte Strategie ausprobieren, die vielleicht einen größeren Reward einbringt ?

$\Rightarrow$ Exploration-Exploitation Dilemma

ORGANIC COMPUTING

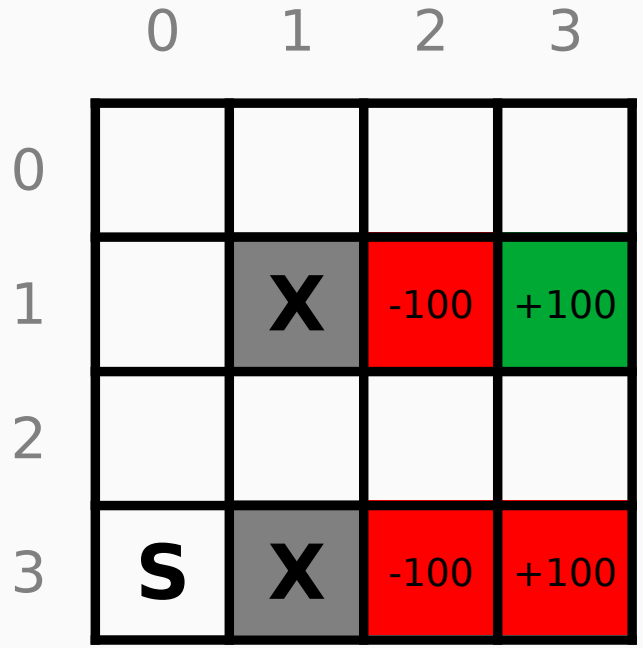
Markov Eigenschaft

Die Zukunft ist unabhängig von der Vergangenheit, wenn man die Gegenwart kennt.



Beispiel

- 4 Aktionen
- Jede Aktion gibt Reward -2
- Von butem Feld führt jede Aktion in Endzustand (Reward 100/-100)



- Mit Chance von 0.8 landet der Agent auf dem Feld auf das er wollte, mit Chance von 0.1 in einer der zwei orthogonalen Richtungen

Markov Entscheidungsproblem

ORGANIC COMPUTING

Notation

- $S = s_1, \ldots, s_n$ ist die Menge aller Zustände (states)
- $A = a_1, \ldots, a_n$ ist die Menge aller Aktionen (actions)
- $T : S \times A \times S \to [0, 1]$ ist die Übergangs-Funktion (transition function). $T(s, a, s')$ beschreibt die Wahrscheinlichkeit das der Agent von Zustand s zu s' wechselt, wenn er Aktion a ausführt.
- $R : S \times A \times S \to \mathbb{R}$ ist die Belohnungs-Funktion (reward function. $R(s, a, s')$ ordnet dem Übergang von Zustand s mittels der Aktion a eine Belohung zu.

ORGANIC COMPUTING

Notation

- $\pi(s)$ ist eine Policy, die dem Zustand s die optimale Aktion zuordnet.
- $\gamma \in [0, 1]$ ist der sogenannte Discount-Faktor. Alle noch erreichbaren Rewards werden nach jeder Aktion mi γ multipliziert. Der Agent soll kürzere Wege bevorzugen. Mit $\gamma = 1$ ist die Länge des Weges egal solange die selbe Belohnung erreicht wird. Mit $\gamma = 0$ ist nur die nächste Belohnung von Bedeutung.

Markov Entscheidungsproblem

ORGANIC COMPUTING

V-Value

V*= Gesamtbelohnung, die der Agent zu erwarten hat, wenn er eine Situation in besagtem Zustand startet, die optimale Aktion ausführt und von dort an optimal handelt

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma * V^*(s')]$$

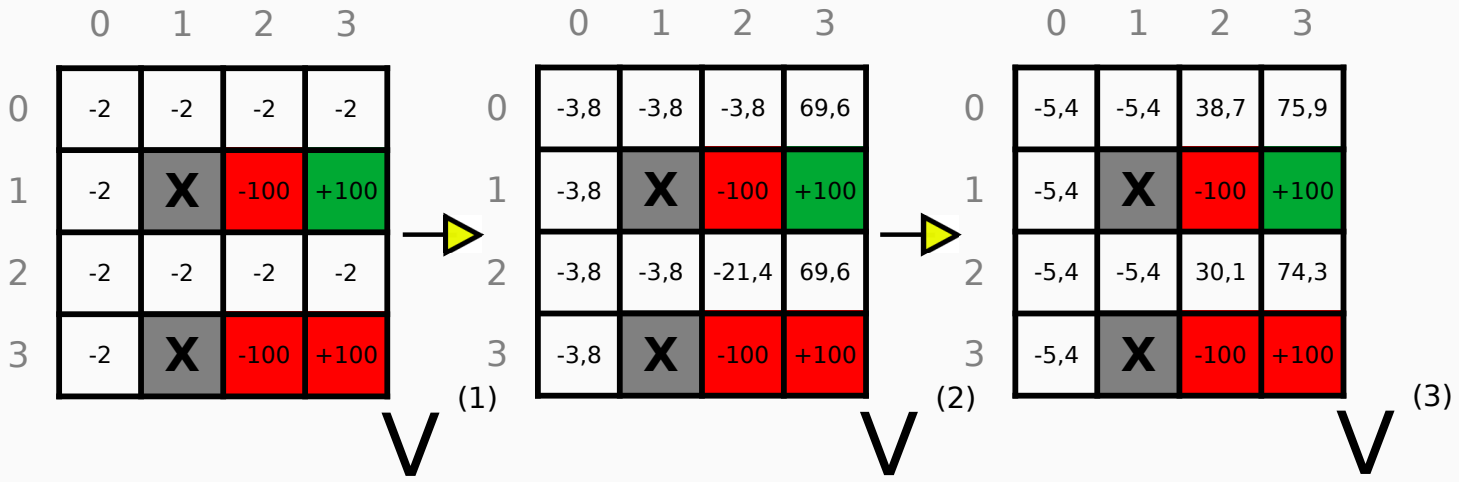
Bedeutung

Wie wertvoll ist der Zustand s

Value Iteration

- zeitl. Begrenzung des MDP
- $V^{(k)}$ = Gesamtbelohnung, nur noch k Schritte übrig

$$V^{(k)}(s) = \max_{a \in A} \sum_{s' \in S} T(s,a,s')[R(s,a,s') + \gamma * V^{k-1}(s')]$$



Value Iteration - State: $(0, 3)$, k=2

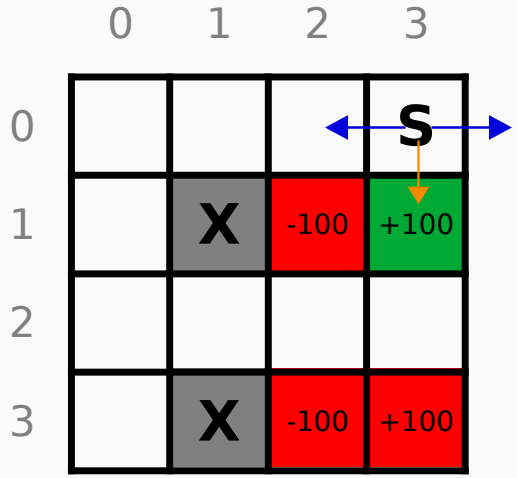
Aktion: South

$$0,8 \times \left(-2 + 0,9 \times 100\right) = 70,4$$

$$0,1 \times \left(-2 + 0,9 \times -2\right) = -3,04$$

$$0,1 \times \left(-2 + 0,9 \times -2\right) = -3,04$$

$$\sum_{s' \in S} \left[70,4; -3,04; -3,04\right] = 69,6$$



Aktion: West $= 5,38$; Aktion: North $= -3,8$; Aktion: East $= 5,38$

$$\max_{a \in A} \left[69,6; 5,38; -3,8; 5,38\right] = 69,6$$

$$V^{(2)}(0,3) = 69,6$$

Value Iteration

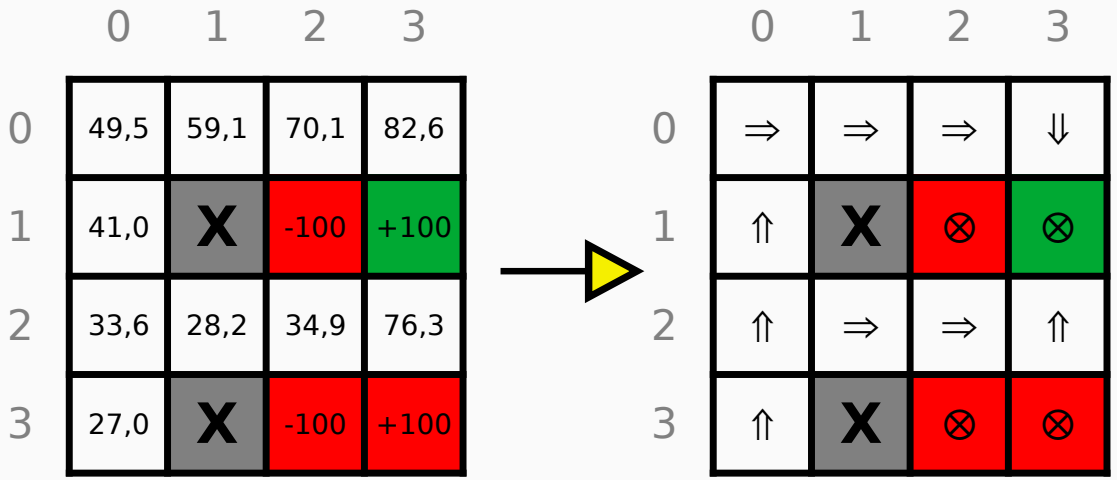
ORGANIC COMPUTING

- max Belohnung des MDP nach oben beschränk
- V^k für $k \to \infty$ konvergiert, da auch mehr Schritte keinen größeren Reward erbringen
- Für geeignet großes k gilt $V^{(k)} = V^* \pm \epsilon$

Policy Extraction

- Mittels der optimalen V-Values V^* lässt sich ein optimale Policy berechnen
- Aktuellem Zustand wird die Aktion zugewiesen, die den erwarteten Reward maximiert

$$\pi^*(s) = \underset{a \in A}{\text{argmax}} \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma * V^*(s')]$$



Q-Learning

Unvollständiges Wissen

ORGANIC COMPUTING

- Value Iteration und policy Extraction benötigen Wissen über $T(s, a, s')$ und $R(s, a, s')$
- Agent muss sich selbstständig über Aktionen durch die Zustände bewegen
- nach jedem Übergang bekommt der Agent den Reward von extern mitgeteilt

Temporal Difference Learning

ORGANIC COMPUTING

- zufällige gewählte V-Values zu Beginn $\rightarrow$ iterativ verbessern
- Nach jeder Aktion überprüft der Agent, wie sich die erwartete Belohnung im Vergleich zur erhaltenen plus der in Aussicht stehenden erwarteten Belohnung verhält und passt die V-Values geringfügig an.

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$$

ORGANIC COMPUTING

Q-Values

$Q(s, a)$ ist die Qualität der Aktion a in Zustand s

$$Q^*(s, a) = \sum_{s' \in S} T(s, a, s')[R(s, a, s') + \gamma V^*(s')]$$

explizites Aktualisieren der Policy kann entfallen, wenn sie implizit mitgelernt wird:

$$V(s) = \max_{a \in A} Q(s, a)$$

$$\pi(s) = \operatorname{argmax}_{a \in A} Q(s, a)$$

ORGANIC COMPUTING

Q-Learning

= Temporal Difference Learning auf Q-Values

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)]$$

Q-Tabelle

ORGANIC COMPUTING

- Q-Funktion als Tabelle
- Für jeden Q-Wert einen Eintrag
- initialisieren der einzelnen Werte (zufällig, mit max Reward oder mit 0)

States \ Actions	NORTH	EAST	SOUTH	WEST
(3,0)	31	26,4	25,7	26,4
(2,0)	37,6	28,1	26,5	32,3
(2,1)	28,1	32,2	28,1	31,2
(2,2)	-69,9	39,6	-69,9	1,5
⋮	⋮	⋮	⋮	⋮

ϵ-greedy Exploration

ORGANIC COMPUTING

- Explorations Strategie
- Hyperparameter $0 \leq \epsilon \leq 1$
- bei jedem Schritt wählt der Agent mit einer Wahrscheinlichkeit von ϵ eine zufällige Aktion
- ϵ nimmt über die Trainingsdauer ab

Pseudocode

Algorithm 1: Q-Learning

Initialize $Q(s, a)$ arbitrarily;
repeat
    Initialize s;
    repeat
        Chose a from s using policy derived from Q (ϵ-greedy);
        Take action a, observe r, s';
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a \in A} Q(s', a) - Q(s, a)]$;
        $s \leftarrow s'$
    until *s is terminal*;
until *Q is converged*;

Aufgaben

Aufgabenblätter

ORGANIC COMPUTING

- Werden nach dem Termin auf Digicampus hochgeladen
- Abgabe bis Sonntag 24:00 Uhr (Vor nächstem Vorlesungstermin)

Quellen

Bilder

1. https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe
2. https://github.com/deepmind/pysc2
3. https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/19564/versions/1/screensh
4. https://2s7gjr373w3x22jf92z99mgm5w-wpengine.netdna-ssl.com/wp-content/uploads/2018/02/reinforcement-learning.png
5. https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690
6. https://hciweb.iwr.uni-heidelberg.de/system/files/private/downloads/541645681/damm learning-report.pdf