Aufgabe 25

(a) (Optimale Bewertungsfunktion zur Bestimmung des kürzesten Pfades)
Wir betrachten einen MDP $(\mathcal{S}, \mathcal{A}, R, \mathbb{P}, \gamma)$ für das Modell eines Spielfelds wie in A23, d.h. der Zustandsraum $\mathcal{S}$ besteht aus den betretbaren Feldern (sowie einem "game over"-Zustand), und der Aktionsraum $\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$. Wählen Sie die Übergangswahrscheinlichkeit $\mathbb{P}$, eine Reward-Funktion R und Abschlagsfaktor γ so, dass die optimale Bewertungsfunktion U gleich der (negativen) Länge des kürzesten Pfades vom aktuellen Zustand zu einem der Exit-Felder ist, und begründen Sie Ihre Wahl kurz.

3	-3	-2	-1	EXIT 0
2	-4		-1	EXIT 0
1	-4	-3	-2	-1
	1	2	3	4

Bestimmen Sie nun (analytisch) für Ihre Wahl die ersten 5 Iterierten U_i, $i = 0, 1, \dots 4$ des Value-Iteration-Algorithmus, wobei anfänglich

$$U_0(s) = \begin{cases} R(s) & s = \text{Exit-Feld oder "game over"} \\ -\infty & \text{sonst} \end{cases}$$

[4 Punkte]

(b) (Alternative Variante der Reward-Funktion)
In der Literatur finden sich Varianten von MDPs, bei denen die Reward-Funktion R zusätzlich von der gewählten Aktion abhängt, also $R(s, a)$. Stellen Sie für diesen Fall eine Bellmann-Gleichung für die Bewertungsfunktion U auf. Wie lässt sich eine solche MDP-Variante zu einem "konventionellen" MDP transformieren (d.h. die Reward-Funktion hängt nur vom aktuellen Zustand ab), so dass die optimale policy unverändert bleibt? [2 Punkte]

Aufgabe 26 (Policy Gradients)

In `pg_maze.zip` auf der Webseite zur Vorlesung findet sich die Vorlage einer "Policy Gradients"-Implementierung[1], mit Anwendung auf das vereinfachte Pac-Man-Spiel in A24. Die policy function π wird hierbei durch ein neuronales Netzwerk beschrieben (siehe `policy_net.py`). Der eigentliche Algorithmus, eine Variante von REINFORCE, findet sich in `pg.py`.

(a) Machen Sie sich mit dem Code in `pg.py` und `policy_net.py` vertraut, und erklären Sie kurz, wie der Gradient $G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$ umgesetzt wird, wobei

$$G_t = \sum_{t'=t}^{\infty} \gamma^{t'-t} R(s_{t'})$$

der "cumulative discounted reward" ab Zeitschritt t ist. Wie realisiert das Programm den Logarithmus in $\log \pi_\theta(a_t|s_t)$? Wieso ergibt es Sinn, die Werte $(G_t)_{t=0,\dots,T}$ (jeweils für einen Spieldurchlauf, wobei T der Spielende-Zeitpunkt) vor dem Gradientenschritt auf Mittelwert 0 und Standardabweichung 1 zu normalisieren? [3 Punkte]

(b) Implementieren Sie die Funktion `discounted_rewards` innerhalb von `pg.py`. Lassen Sie nun `run_ghost_maze_policy_gradients.py` laufen, und geben Sie die abschließende Ausgabe (ab `episode 100000 completed`) an. [3 Punkte]

Hinweis: Das Programm protokolliert den (über mehrere Spielsimulationen gemittelten) Wert G_0 als "running mean" mit; zum Vergleich wird der numerisch exakte Erwartungswert für die optimale policy, d.h. die (über alle Zustände gemittelte) value function $U(s)$, mittels Policy-Iteration berechnet und unter "optimal value function average" ausgegeben.

[1] siehe auch https://karpathy.github.io/2016/05/31/rl

Tutoraufgabe 13 (Partially Observable Markov Decision Process)
Ein "partially observable Markov decision process" (POMDP) modelliert eine "partielle Sichtbarkeit" des Zustands, d.h. der Agent kennt den tatsächlichen Zustand s_t zum Zeitpunkt t nicht sicher, sondern lediglich eine "Evidenz" $e_t \in \mathcal{E}$, die einer bedingen Wahrscheinlichkeitsverteilung $\mathbb{P}_{\mathsf{evi}}(e_t|s_t)$ folgt. Formal besteht ein POMDP neben der endlichen "Evidenzmenge" $\mathcal{E}$ und $\mathbb{P}_{\mathsf{evi}}$ aus denselben Elementen wie ein MDP, d.h. er wird durch das Tupel $(\mathcal{S}, \mathcal{A}, \mathcal{E}, R, \mathbb{P}, \mathbb{P}_{\mathsf{evi}}, \gamma)$ beschrieben.

Ein POMDP kann formal wiederum als MDP aufgefasst werden, wobei jetzt der Zustand des Agenten ein sogenannter "belief state" b_t ist, d.h. eine diskrete Wahrscheinlichkeitsverteilung über die möglichen Zustände, in denen sich der Agent befinden könnte ($b_t(s) \geq 0$ für alle $s \in \mathcal{S}$, $\sum_{s\in\mathcal{S}} b_t(s) = 1$)[2].

Geben Sie den Übergang bzw. die Übergangswahrscheinlichkeit vom aktuellen belief state b_t in einem Zeitschritt zum nächsten an. Wieso lassen sich die bisherigen Iterationsalgorithmen (wie Value-Iteration) nicht mehr ohne Modifikation einsetzen?

[2] Der Zustandsraum für den belief state ist somit kontinuierlich, so dass es sich streng genommen um keinen MDP gemäß der bisherigen Definition mehr handelt.