Grundlagen der Intelligenten Systeme (WS2024/25)

Programmierübung Bonus

Deadline für die Einreichung: 14.01.2025
Anzahl der Aufgaben: 1
Maximal erreichbare Punktezahl: 100 Punkte

Allgemeines

- Diese ist eine von zwei Programmierübungen in *Grundlagen der intelligenten Systeme*. Sie können bis zu **100 Punkte** auf diese Übung erhalten.
- Die Deadline für die Einreichung dieser Programmierübung ist der **14.01.2025** und ist nicht verlängerbar. Für verspätete Abgaben wird Ihre erreichte Punktezahl um 25 % pro angefangenen Tag reduziert. Konkret bedeutet dies Folgendes: Angenommen, Sie machen eine um t Tage verspätete Abgabe, die wenn Sie zeitgerecht erfolgt wäre, mit p Punkten beurteilt wäre. In diesem Fall würden Sie $\max\{0, p \cdot (1 - 0.25t)\}$ Punkte für Ihre Abgabe erhalten.
- Bei Fragen und Problemen kontaktieren Sie uns rechtzeitig—primär über Moodle [1].
- Lösen Sie die Programmierübungen in **Python 3**. Verwenden Sie **keine** Bibliotheken die nicht explizit erlaubt ist.
- Stellen Sie sicher, dass Ihr Code ausführbar ist und das geforderte Interface implementiert. Jupyter Notebooks werden nicht akzeptiert.
  ⚠ Wenn Ihr Code nicht ausführbar ist oder die angeführten Spezifikationen nicht einhält, erhalten Sie für Ihre Abgabe Null Punkte.
- Lösen Sie die Programmieraufgaben eigenständig. Wir verwenden Tools wie Turnitin zur Detektion von Plagiaten.
  ⚠ Wenn wir ein Plagiat entdecken, werden Sie in der Lehrveranstaltung mit einem "X" beurteilt.
- Bennen Sie Ihre ausführbaren Python-Skripte mit `aufgabeZ.py`, wobei `Z` die Nummer der gelösten Aufgabe ist. Zippen Sie alle Ihre Skripte und benennen Sie die Datei `matrikelnummer_nachname_vorname.zip`, wobei Sie `matrikelnummer` durch Ihre Matrikelnummer, `nachname` durch Ihren Nachnamen und `vorname` durch Ihren Vornamen ersetzen. **Halten Sie sich nicht an diese Benennungsregeln, ist ihr Code nicht ausführbar.**

1 Lernziele

Ziel dieser Programmieraufgabe ist es, erste Erfahrungen in der Entwicklung von logischen, probabilistischen, oder auch Reinforcement Learning-basierten Agenten zu erwerben. Welche Art von Agent Sie konkret entwickeln, ist hierbei Ihnen überlassen.

2 Aufgaben

Lösen Sie alle Aufgaben durch das Erstellen von Python-Skripten. Sie können folgende Bibliotheken verwenden: `numpy`, `os`, `matplotlib`, `sklearn`, `pillow`, `tqdm`.

Voraussetzungen. Sie benötigen folgende Python Bibliotheken: `numpy`, `pillow`, `tqdm`, `pygame` (nur für die graphische Darstellung). Abhängig von Ihrem Ecosystem, können diese etwa mittels `pip install library_name` installieren.

Aufgabe 1: Intelligente Agenten

(Maximal erreichbare Punktezahl: 100 Punkte)

Ziel dieser Aufgabe ist es einen intelligenten für die Wumpus-Welt zu entwickeln der möglichst hohe Rewards erreicht. Hierfür stellen wir Ihnen die Wumpus-Welt als Python-Code zur Verfügung (`wumpus.py`). Der Code ermöglicht es Ihnen die Wumpus-Welt zu instantiieren und mit ihr zu interagieren. Konkret können Sie aus der Datei `wumpus.py` die Klasse `Wumpus` importieren und davon Instanzen erzeugen. Diese stellen folgende Funktionen zur Verfügung:

- `init(seed=2024, size=(4,4), p_pit=0.2, Tmax=50)`
  Konstruktor. Initalisiert einen Zufallszahlengenerator für die Erzeugung von Wumpus-Welten mit Seed `seed` (Default-Wert `2024`). Darüber hinaus kann die Größe der Wumpus-Welt mittels `size` (Default-Wert `(4,4)`), die Wahrscheinlichkeit `p_pit` das ein Feld eine Falltür enthält (Default-Wert `0.2`), sowie die maximale Anzahl an Interaktionsschritten `Tmax` bevor dem Agent in der Höhle die Batterie ausgeht (Default-Wert `50`)[1] übergeben werden. Für die Lösung der Aufgabe müssen diese Default-Werte nicht angepasst werden.

- `reset()`
  Erstellt eine neue Wumpus-Welt und setzt den Agenten zurück in seinen Anfangszustand. Diese Funktion muss vor der ersten Interaktion mit einer Wumpus-Welt ausgeführt werden.

[1] Geht dem Agenten in der Höhle die Batterie aus, erhält der Agent einen Reward von -1000

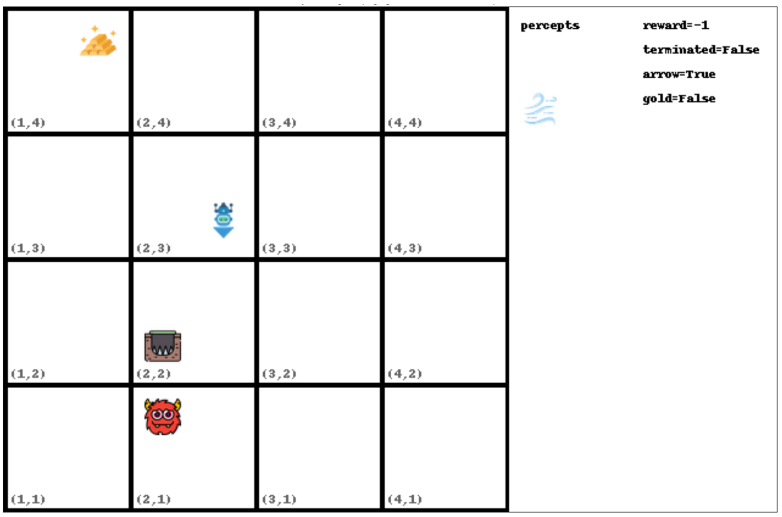

Figure 1: Beispielhafte Wumpus-Welt

- `render()`
  Erzeugt eine graphische Darstellung der Wumpus-Welt. Die Darstellung umfasst die Position des Agenten, des Wumpus, der Falltüren, des Goldes (auch wenn diese nicht direkt vom Agenten beobachtar sind), die Beobachtungen des Agenten, sowie Informationen über den erhaltenen Reward, ob der Agent noch einen Pfeil hat, usw. Diese kann beispielsweise mittels der Funktion `imshow` von `mathplotlib` dargestellt werden. Ein Beispiel ist in Abbildung 1 dargestellt.

- `step(action)`
  Führt die Aktion `action` in der Wumpus-Welt aus. Als Ergebnis erhält man ein Tuple `(percept, reward, terminated, info)`. Hierbei:

  - `percept` ist ein fünf dimensionaler Vektor, der die Beobachtungen Gestank / Luftstoß / Glitzern / Stoß / Schrei in Form von Binärwerten enthält (1 bedeutet, die entsrepchende Wahrnehmung wurde gemacht, 0 bedeutet keine entsprechende Wahrnehmung wurde gemacht).

  - `reward` ist der Reward für die letzte durchgeführte Aktion.

  - `terminated` ist eine Binärvariable, die anzeigt ob die Interaktion beendet ist (der Agent ist gestorben oder die Maximalanzahl an Interaktionen ist verstrichen).

  - `info` kann zusätzliche Statusinformationen enthalten, kann aber für die Lösung der Aufgabe ignoriert werden.

Testen der Wumpus-Welt. Sie können mit der Wumpus-Welt interagieren, indem Sie `python wumpus.py` ausführen. Mittels Tasten kann Ihr Agent gesteuert werden:

- `up` Bewegung in Blickrichtung (wenn möglich)
- `left` Drehung um 90 Grad nach links.
- `right` Drehung um 90 Grad nach rechts.
- `t` Abfeuern eines Pfeils, wenn vorhanden.
- `g` Aufheben des Goldes vom Boden auf, falls möglich.
- `c` Klettern aus der Höhle, wenn möglich.
- `r` Reset der Weumpus-Welt (erstellt eine neue Welt).
- `q` Beenden der Interaktion.

Implementierung eines Agenten. Zur Lösung des Problems müssen Sie einen intelligenten Agenten entwickeln. Hierfür müssen Sie eine Klasse mit Namen `Agent` erstellen. Diese muss die folgenden zwei Funktionen implementieren:

- `get_action(percept, reward)` — gibt die nächste auszuführende Aktion zurück. Hierbei ist `percept` die Wahrnehmung des Agenten, `reward` der letzte erhaltene Reward.
- `new_episode()` — teilt dem Agenten mit, dass eine Interaktion mit einer neuen Wumpus-Welt beginnt.

Beispiel. Zwei Beispiele für Agenten sind in der Datei `interact.py` implementiert.

Abgabe. Die Datei `aufgabeZ.py` muss den von Ihnen implementierten Agenten inklusive etwaiger notwendiger Imports beinhalten. Für die Evaluierung importieren wir Ihren Agenten, instanzieren ihn und testen ihn auf mehreren Wumpus-Welten. Hierbei setzen wir ein Zeitlimit von 10 Minuten für 1000 Wumpus-Welten.

Teilaufgabe 1: Implementieren eines Agenten.

[100 Punkte] Implementieren Sie einen Agenten gemäß den oben angeführten Vorgaben. Ihr Agent wird auf 1000 zufällig generierten Wumpus-Welten getest und der

mittlere erreichte Reward $\bar{R}$ ermittelt. Sie erhalten p Punkte auf Ihre Abgabe:

$$p = \min\{100, 100 \cdot \frac{\bar{R} - 100}{600}\}.$$

Ist $p < 0$, erhalten Sie 0 Punkte. Ist Ihr Agent nicht ausführbar, erhalten Sie 0 Punkte für Ihre Abgabe.

Die erhaltenen Punkte werden dann in Bonuspunkte konvertiert. Der Schlüssel hierfür wird noch bekanntgegeben.

Hinweis: Orientieren Sie sich für die Implementierung Ihres Agenten an `interact.py`.

3 References

[1] *Grundlagen der intelligenten Systeme, Moodle Page.* URL: `https://moodle.univie.ac.at/course/view.php?id=437205`.