Huiswerk Opdracht 1

Deze huiswerkopdracht bevat vragen en programmeeropdrachten. Bij elke opdracht staat het aantal punten dat er voor te verdienen is. Iedere student moet de opdrachten zelfstandig uitvoeren. Volg de volgende stappen bij het inleveren van de opdrachten:

- Maak een map met daarin:
  - Een PDF-document met alle vragen en antwoorden.
  - Je Python-bestanden en evt. bijgeleverde scripts (die zullen nodig zijn om de code uit te voeren).
- Comprimeer de map als een zip-bestand met de naam `HW1-{voornaam}_{achternaam}.zip`.
- Stuur het bestand op naar abe_vos@msn.com met het onderwerp `HW1 {voornaam} {achternaam}`.

De deadline is donderdag 12 mei om 23:59. Vragen over het huiswerk kun je sturen naar abe_vos@msn.com.

Gridworld (2 punt)

In deze oefening gaan we een beleidsfunctie evalueren. De omgeving die we gebruiken staat in gridworld.py en is gebaseerd op een omgeving van OpenAI Gym.

We kunnen als volgt een instantie van de omgeving maken:

```
from gridworld import Gridworld

env = Gridworld()
```

We kunnen ook een simulatie uitvoeren:

```
# Reset de omgeving en krijg de eerste observatie.
state = env.reset()

# Print de huidige staat.
env.render()

# We kunnen vier acties uitvoeren:
#  0: omhoog
#  1: rechts
#  2: omlaag
#  3: links
new_state, reward, done, _ = env.step(0)

# Print de nieuwe staat.
env.render()
```

Bestudeer de code in `gridworld.py` en omschrijf de spelregels van deze gridworld. Is de omgeving stochastisch of deterministisch?

Policy Evaluation (6 punten)

In deze oefening gaan we Policy Evaluation implementeren. We hoeven hiervoor geen simulaties van de omgeving uit te voeren. We willen we de omgevingsdynamieken gebruiken. Deze zijn normaal gesproken niet beschikbaar, maar voor deze gridworld zijn ze opgeslagen in $env.P$.

We beginnen met een beleid waarin voor elke staat iedere actie met gelijke kans gekozen kan worden, oftewel de kans voor elke actie is 0.25. We implementeren dit beleid als een matrix met numpy:

```
import numpy as np

policy = np.ones((25, 4)) / 4
```

Schrijf een functie die Policy Evaluation uitvoert voor een gegeven beleid. Hieronder volgt de opzet voor de functie:

```
def policy_eval(policy, env, discount_factor=0.9, theta=1e-5):
    V = np.zeros(env.observation_space.n)
    delta = np.inf

    while delta > theta:
        # SCHRIJF HIER JE CODE
        pass

    return np.array(V)
```

Voer de functie uit voor het beleid die altijd gelijke kansen aan elke actie geeft. Rapporteer de waardes van de waarde functie in een tabel van 5 rijen en 5 kolommen.

Omschrijf wat er gebeurt als je `discount_factor=1` gebruikt.

Policy Iteration (4 punten)

Nu we Policy Evaluation hebben geimplementeerd, kunnen we Policy Iteration implementeren.

```
def policy_improvement(env, discount_factor=0.9):
    nS = env.observation_space.n
    nA = env.action_space.n
    policy = np.ones((nS, nA)) / nA
    policy_stable = False

    while not policy_stable:
```

```
        V = policy_eval(policy, env, discount_factor)

        policy_stable = True

        for state in range(nS):
            old_action = np.random.choice(nA, p=policy[state])

            # BESCHRIJF DE NIEUWE ACTIE
            new_action = None

            policy[state] = 0
            policy[state, new_action] = 1

            if old_action != new_action:
                policy_stable = False

    return policy, V
```

Update de bovenstaande code zodat het Policy Improvement algoritme werkt. Maak een afbeelding die voor elk vlak in het raster een pijl laat zien die overeenkomt met de optimale actie in die staat. Toon de optimale waarde functie.

Value Iteration (6 punten)

Nu implementeren we Value Iteration. Een uitgebreide omschrijving van Value Iteration is te vinden in Sutton & Barto op pagina 101 (figuur 4.5).

```
def value_iteration(env, theta=1e-5, discount_factor=1.0):
    V = np.zeros(env.observation_space.n)
    policy = np.ones((nS, nA)) / nA

    # SCHRIJF HIER JE CODE

    return policy, V
```

Laat de resultaten van de geleerde waardefunctie en beleid weer zien.

Omschrijf welk algoritme (Policy Iteration of Value Iteration) het meest efficient is en waarom?