Practice 2
Dynamic Programming
Policy Iteration and Value Iteration

Learning Objectives

The objective of this Practice is to develop a Policy iteration and a Value iteration in the Russell's Grid Universe that we developed in Practice 1 ([RN10]). In this Practice we will learn how to apply the two basic reinforcement loops for Dynamic Programming in this world. These Loops can be applied to any other world (Activity 3).

I provide with the Practice_2_solved.ipynb file. You don't need to do activity 1 and 2, but you must run the file Practice_2_solved.ipynb in your environment. You need to answer the questions in Activity 3 and solve the FrozenLake using dynamic programming (follow the hint).

1 Files for this Practice

```
015_DProgramming_Russells_World-BASELINE.ipynb
015_DProgramming_Taxi.ipynb
```

Background on Rusell's Grid World

Russell's World is a grid-based environment where an agent must navigate from a start position to a goal position while avoiding obstacles and possibly collecting rewards. The environment is defined by:

- **Observation space (S)**: The set of all possible positions in the grid.
- **Action space (A)**: The set of possible moves (e.g., left, right, up, down).
- **Rewards (R)**: Values received after moving from one state to another.
- **Transitions (T)**: The probabilities of moving from one state to another given an action.
- **Agent** starts in position (1,1)

This environment is based on a universe of a 3x4 board with a set of cells and two terminal states. Beginning in the start state, it must choose an action at each time step. The interaction terminates when the agent reaches one of the goal states ($+1$ or -1), one is a positive end the other a negative one.

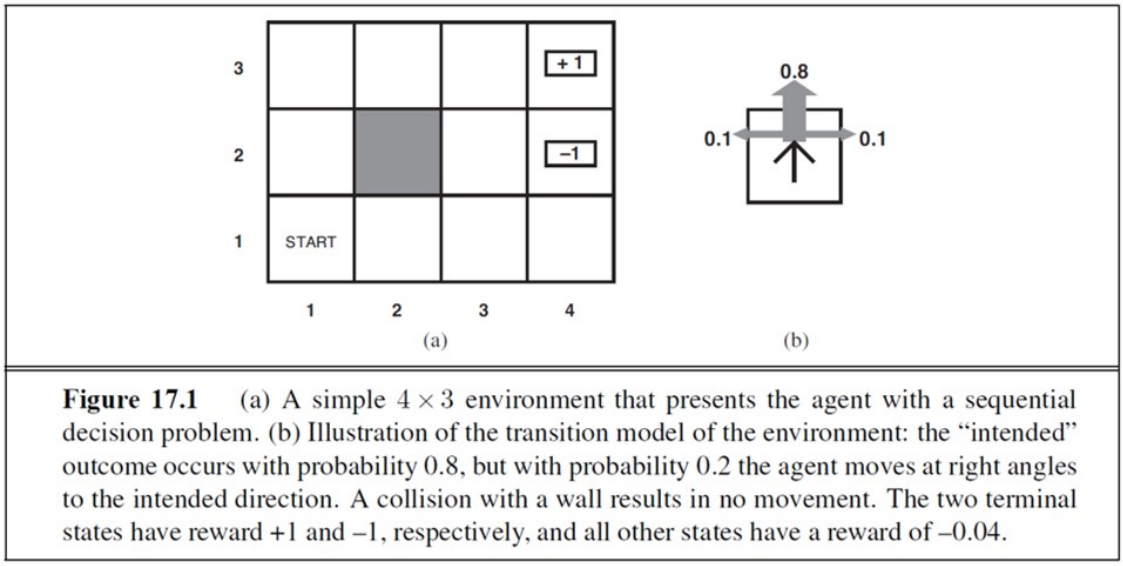

Figure 17.1 (a) A simple 4×3 environment that presents the agent with a sequential decision problem. (b) Illustration of the transition model of the environment: the "intended" outcome occurs with probability 0.8, but with probability 0.2 the agent moves at right angles to the intended direction. A collision with a wall results in no movement. The two terminal states have reward +1 and –1, respectively, and all other states have a reward of –0.04.

Figure 1: Image from the book [RN10], Chapter 17

Unwrapped environment

When you see the term "unwrapped environment", it refers to accessing the raw, original environment by bypassing any wrappers that have been applied to it. This allows direct access to internal variables, such as the policy. In the assignments for Taxi and Frozen Lake, you will need to use the unwrapped environment to access the policy from the program.

2 Activities

The activities in this Practice are based in the Russell's Grid universe described in Practice 1.

The RussellGrid world environment is included in the code. You can have a look at the implementation in the notebook.

We will perform 3 activities in this practice. First we will implement the Policy evaluation loop, then the Value evaluation loop and finally we will apply them to a different environment, in this case to the Frozen Lake environment.

2.1 Analyzing the data structures to develop RL algorithms

In the previous practice we created the environment RussellGrid, we did not put lots of attention on how it was done inside, but in this practice we need to understand it in detail.

The environment RussellGrid has been created from scratch making it compatible with the Gymnasium standard environment definition.

To do this exercise we need to understand how the environment is implemented in python and which are the main data structures we will use, mainly the **Value vector** $V(s)$ and the **Policy** π.

How to represent Vector $V(s)$ in Python

The environment has an observation space of 12 states. There is, in the gymnasium environments a method that tells us the observation space dimension. A way to invoke the dimension is:

```
env.observation_space.n
```

One of the issues we will find when we work with this environments is that we may have 12 states, but we need to manage them in 2D, as they are in a matrix of 3rows x 4 columns. This may generate some confusion but you need to have this in mind when you work with the data structures.

The **Value Vector** then will have as many positions as value spaces. for this example you only need to take into account that the Value vector has 12 positions (from 0 to 11) and those positions correspond to two dimensional positions. See Gigure 2

0 [0,0]	1 [0,1]	2 [0,2]	3 [0,3]
4 [1,0]	5 [1,1]	6 [1,2]	7 [1,3]
8 [2,0]	9 [2,1]	10 [2,2]	11 [2,3]

Figure 2: Coordinates in RussellGrid Envionment

The Policy is a Dictionary P

This is the critical structure in Reinforcement Learning. The Policy tells us the probabilities between each state and the possible transition steps. We need to understand how this dictionary is built.

The Policy in Gymnasium is a dictionary in the class that can be invoked as:

```
env.P
```

For intance, this invokation will generate this output for the RussellGrid environment

```
{0: {0: [(0.8666666666666667, 0, -0.04, False),
   (0.06666666666666667, 1, -0.04, False),
   (0.06666666666666667, 4, -0.04, False)],
  1: [(0.8, 1, -0.04, False),
   (0.13333333333333333, 0, -0.04, False),
   (0.06666666666666667, 4, -0.04, False)],
   ...
```

The dictionary has several positions:

```
P[s][a][(s', prob(s|s'), reward, final)]
```

For instance, in the previous listing, the interpretation is: From state 0 with action 0 there is a 0.86 probability to end in state 0, with a reward of -0.04 and is not a final state. From state 0 with action 1 there is a 0.06 probability to end in state 1, with a reward of -0.04 and is not a final state, so on so forth. The dictionary has this structure then

```
P[0][1][(0, 0.86, -0.04, False)]
       [(1, 0.06, -0.04, False)]
       [(4, 0.06, -0.04, False)]
                  ...
```

2.2 Implementing the Policy Iteration Loop - ACTIVITY 1

We will implement some loops in this practice. The evaluation and Policy evaluation loops described in class which are introduced in the book [SB18]. Follow the instructions in detail and you will manage to finish this activity without any issues.

We will implement the Policy evaluation loop in the notebook it is marked as [TO CODE Activity 1]

Iterative Policy Evaluation Algorithm

Given:

- A Markov Decision Process (MDP) with states S, actions A, state transition probabilities $P(s'|s,a)$, and reward function $R(s,a)$.
- A policy $\pi(a|s)$ that specifies the probability of taking action a in state s.

Initialize the value function $V(s) = 0 \quad \forall s \in S$.

repeat

$\Delta \leftarrow 0$

for each $s \in S$

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{a \in A} \pi(a|s) \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$

$\Delta \leftarrow \max(\Delta, |v - V(s)|)$

until $\Delta < \theta$ {where θ is a small positive threshold}

Output: The value function $V(s)$ for the policy π.

The first step is to create a function called policy evaluation that performs the algorithm.

```
def policy_evaluation(env, policy, theta=1e-8, gamma=1.0):
    V = np.zeros(env.observation_space.n)
    while True:
        delta = 0
        for s in range(env.observation_space.n):
            if s == env.invalid_position[0] * env.ncol + env.
               invalid_position[1]:
                continue
            v = 0
            for a, action_prob in enumerate(policy[s]):
                for prob, next_state, reward, done in env.P[s][a]:
                    v += action_prob * prob * (reward + gamma * V[
                       next_state] * (not done))
            delta = max(delta, np.abs(v - V[s]))
            V[s] = v
        if delta < theta:
            break
    return V
```

Let's try to explain the code.

- **V** Value Function is a numpy array indexed from 0 to 11.
- **delta**: An auxiliary variable for convergence
- **S** is all the states. for each s ∈ S is a loop on all states

- **v** auxiliary variable to calculate the iteration of the $\sum$ line
- for every state $\leftarrow$ for s in range all states
- for every prob and every action for a state
- For a given state and action, env.P[state][action] gives you a list of possible transitions., each state P is a tuple of (probability, next_state, reward, done)
- Bellman equation

Complete Policy Iteration (Sutton)

1. Initialization

   $V(s) \in \mathrm{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation

   Given:
   - A Markov Decision Process (MDP) with states S, actions A, state transition probabilities $P(s'|s,a)$, and reward function $R(s,a)$.
   - A policy $\pi(a|s)$ that specifies the probability of taking action a in state s.

   Initialize the value function $V(s) = 0$ for all $s \in S$.
   repeat
   for each $s \in S$
   $v \leftarrow V(s)$
   $V(s) \leftarrow \sum_{s',r} p(s', r|s, a)\,[r + \gamma V(s')]$
   $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ {where θ is a small positive threshold}
   Output: The value function $V(s)$ for the policy π.

3. Policy Improvement

   Input: A policy π and value function V.
   $policy\_stable \leftarrow True$
   for each $s \in \mathcal{S}$
   $a \leftarrow \pi(s)$
   $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma V(s')]$
   if $a \neq \pi(s)$ **then**
   $policy\_stable \leftarrow False$
   Output: Improved policy π

This function corresponds with the inner loop of the policy evaluation algorithm.

It looks impossible but let's clarify the exercise by building a small dictionary that will allow you to perform the exercise.

Dictionary of elements in the algorithm. You'll find some hints to build the iteration loops

- S : env.S
- γ : $gamma = 0.99$
- $V(s)$: $V[s]$
- Initialise $V(s)$ for all s: `V = np.zeros(env.num_states)`

The policy iteration loop is as follows, and the only you have to do.

Policy Iteration

POLICY ITERATION CODING PROPOSAL

Input: env, theta, gamma.
policy $\leftarrow$ np.ones([obs.space, env.action.space])/env.action.space
i = 0
repeat
  $v \leftarrow policy\_evaluation(env, policy, theta, gamma)$
  $new\_policy = policy\_improvement(env, v, gamma)$
  $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
  $policy = new\_policy$
  $i = i + 1$
until $\Delta < \theta$
Output: policy π, V

HINT

$\Delta \leftarrow np.all(np.abs(policy - new\_policy))$

2.3 Implementing the Value Iteration Loop - ACTIVITY 2

Value Iteration (Sutton)

Initialize the array V arbitrarilty (e.g., $V(s) = 0$ for all $s \in S$).
repeat
  $\Delta \leftarrow V(s)$
  for each $s \in S$
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\,[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$ {where θ is a small positive threshold}
Output: A deterministic policy, π, such that
$\pi(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma V(s')]$

2.4 Questions - ACTIVITY 3

When you have your code ready run it and try to understand the results Ask yourself the following questions:

1. Which strategy converges faster, Value Iteration or Policy Iteration?
2. Are both optimal policies similar? Why?
3. You can see how the Value function generates easily the optimal policy function. Would it be possible to generate the Value function from the optimal policy? Why?
4. If you modify the values gamma and theta, do you find any improvement? (remember γ is used to delay rewards and θ is just a threshold).

2.5 Dynamic Programming Frozen Lake environment Loop - ACTIVITY 4

If you followed the standard naming and procedures your loops work for any Gymnasium discrete environment. Try to implement your loops in the Frozen Lake environment. You can verify the information about the Gymnasium package in [Far24].

Hint: Use as a template the notebook

```
015_DProgramming_Taxi.ipynb
```

That performs the loops for the Taxi environment.

3 Deliverables and submission

Submit two notebooks

- Notebook with the Dynamic Programming Russell's World
- Notebook with the Frozen Lake with Dynamic Programming

Respond to the questions in Activity 3 in the first Notebook

Use the naming convention for the notebooks

There is a Gradebook Entry. Use a zip file with your name to group all the documents.

Naming Convention

Create two notebooks with the names

Include your name in naming the notebooks, for example for a stdent with the name Joseph Rodriguez do as follows;

```
Practice2-DP_RussellWorld_Joseph_Rodriguez.ipynb
Practice2-DP_FrozenLake_Joseph_Rodriquez.ipynb
```

References

[RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 4th ed. Prentice Hall, 2010. ISBN: 9780136042594.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: `http://incompleteideas.net/book/the-book-2nd.html`.

[Far24] Farama. *Gymnasium package documentation*. `https://gymnasium.farama.org/`. Accessed: 2024-08-09. 2024.