# D7047E Exercise 6
# Reinforcement Learning

## Introduction

The goal of this exercise is to introduce you to reinforcement learning (RL) and hopefully by the end of this lab you have programmed a deep reinforcement learning algorithm that can learn by itself how to play an Atari game. Below is a brief summary of RL but it is recommended to look at the RL content in the last module before doing this lab.

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 [1].

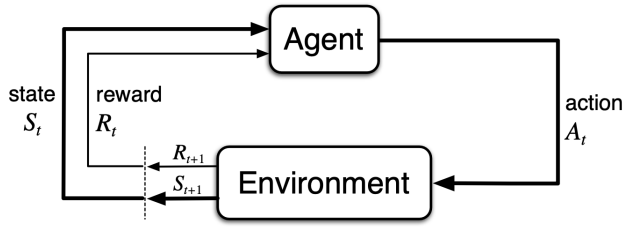


Figure 1: The reinforcement learning loop

Summary of the reinforcement learning loop:

- Observation: The agent observes the current state of the environment. This can include information such as sensor readings, raw data, or any other relevant information that represents the current state of the environment.
- Action: Based on the observed state, the agent selects an action to take. This action is chosen from the set of possible actions that the agent can take in the current state of the environment. The action can be deterministic or stochastic, depending on the algorithm and the problem setting.
- Environment: The agent performs the selected action, and the environment transitions to a new state based on the dynamics of the environment. The environment may also provide the agent with a reward signal that indicates the immediate feedback on the action taken.
- Reward: The agent receives a reward from the environment, which is a scalar value that reflects the desirability of the outcome of the action taken. The reward can be positive, negative, or neutral, and it provides feedback to the agent on how well it is performing

All of these concepts can be modelled by a Markov decision process (MDP) which we will learn more about in task 1. Most of the strategies for learning in reinforcement learning broadly fit under two categories, Q-learning and policy gradient methods. Q-learning is a value-based method that estimates the action-value function and uses the Bellman equation for updates, while policy gradient methods are policy-based methods that directly optimize the policy using gradient-based optimization. Both methods have their advantages and disadvantages and are used in different scenarios depending on the characteristics of the problem at hand. In the practical section we will explore deep Q networks (DQN) further.

# Theoretical Part

## Task 1 - Gridworld

This is Gridworld, a simple environment where you can navigate through a grid using four cardinal directions: North, South, West, and East. Reinforcement learning environments are typically modeled as (MDPs), which is a mathematical framework used to describe decision-making problems in uncertain environments. Before we dive into Gridworld, let's first explore what an MDP is.

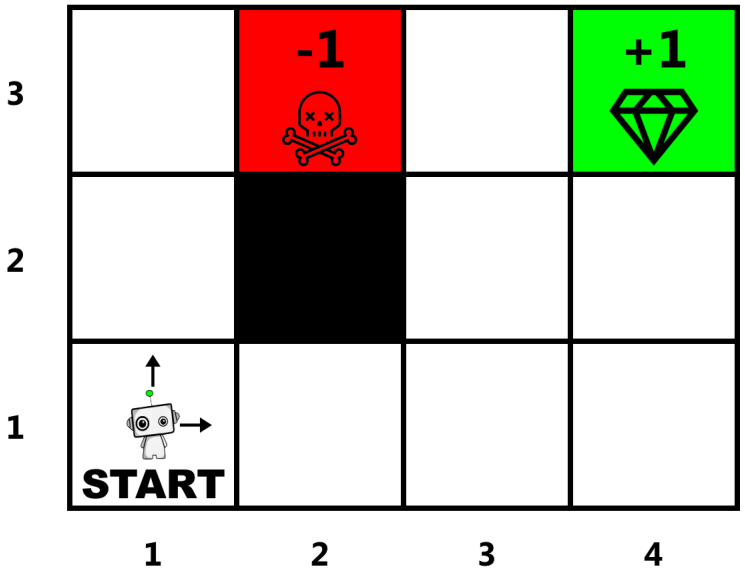


Figure 2: Gridworld

A Markov Decision Process (MDP) is defined by:

- Set of states $S$
- Set of actions $A$
- Transition function $P(s'|s,a)$ is the probability of going to $s'$ given the current state $s$ and action $a$.
- Reward function $R(s,a,s')$ returns the reward after taken action $a$ in state $s$ ending up in $s'$.
- Start state $s_0$
- Discount factor $\gamma$, defines how much we care about immediate reward compared to future reward.
- Horizon H, how long our agent lives.

The goal of a MDP can then be defined by the following equation:

$$max_\pi \mathbb{E}[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1})|\pi] \tag{1}$$

where $\pi$ is our policy. Hence the goal is to find the optimal policy that maximizes our expected reward.

**Task 1.1 Optimal Policy $\pi^*$**

A policy $\pi$ is a mapping from state to action. This means that the optimal policy $\pi^*(s)$ should represent the best action to take for every possible state.

Assume that gridworld is completely deterministic and draw an arrow(s) in each white cell so that your solution represents the optimal policy $\pi^*(s)$.

**Value Function $V$**

A value function in reinforcement learning is a mathematical function that estimates the expected total reward an agent can achieve from a given state, while following a particular policy. It quantifies the desirability or utility of a state in terms of the rewards the agent can expect to receive over time.

**Optimal Value Function $V^*$ definition:**

$$V^*(s) = max_\pi \, \mathbb{E}[\sum_{t=0}^{H} \gamma^t R(S_t, A_t, S_{t+1}) | \pi, s_0 = s] \tag{2}$$

which is the sum of expected discounted rewards when starting from state $s$ and acting optimally.

**Task 1.2**

Assume that the transition function is equal to 1, $\gamma = 0.9$ and $H = 100$. Calculate the following using Figure 2 and Eq 2:

- $V^*(4,3) =$
- $V^*(3,3) =$
- $V^*(2,3) =$
- $V^*(3,1) =$
- $V^*(1,1) =$

**Task 1.3**

Assume that the transition function is equal to 0.8, $\gamma = 0.7$ and $H = 100$. Calculate the following using Figure 2 and Eq 2:

- $V^*(4,3) =$
- $V^*(3,3) =$

Hint: Don't calculate the recursion.

## Task 2 - Questions

**Task 2.1: Explain the exploration vs. exploitation problem in RL**

**Task 2.2: Explain the credit-assignment problem in RL**

**Task 2.3: What is the Markov property?**

**Task 2.4: Motivate whether or not chess satisfies the Markov property**

**Task 2.5: What is the difference between Q-learning and Deep Q-learning?**

# Practical Part

In 2013 Deepmind published *Playing Atari with Deep Reinforcement Learning* where they showed that a reinforcement learning algorithm can learn how to play many different Atari games completely on its own with only the pixel values as input. The algorithm they used is called DQN with Experience Replay, which is what you will implement in this lab. You can watch Demis Hassabis present this in the Deepmind AlphaGo documentary.

**Algorithm 1** Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
**for** episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  **for** $t = 1, T$ **do**
    With probability $\epsilon$ select a random action $a_t$
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  **end for**
**end for**

## Setup

This might not work on all python versions, it has been tested with python 3.9. Therefore you can set up a conda environment with python 3.9 and then activate it:

```
conda create -n ex6 python=3.9

conda activate ex6
```

Then you have to install some libraries that will handle the preprocessing for us.

```
pip install -r requirements.txt
```

## Files in canvas

- `example.py` - To test if your installation is working, if it is you should see a game of breakout running a policy that selects random actions. (Read the warning if you have an M1 Mac.)
- `hyperparams.py` - contains all the default hyperparameters.
- `dqn.py` - starting code to solve task 1.1 and 1.2

### Tips

- Use the gym documentation for a deeper understanding of the games and how the library functions.

# Breakout

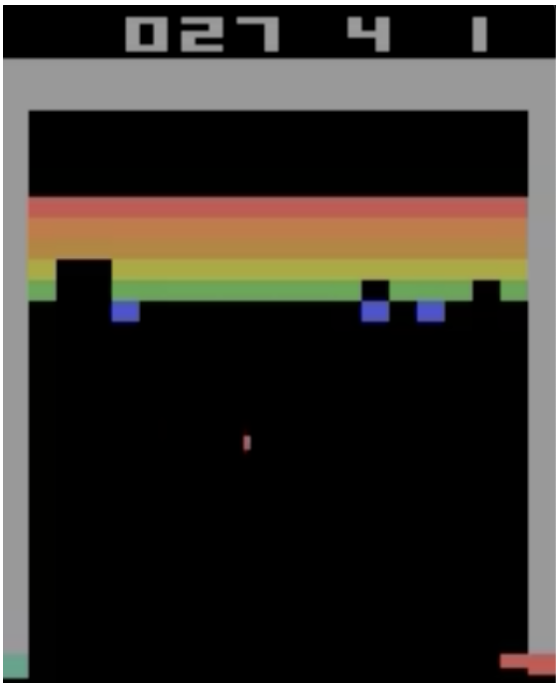


Figure 3: Arcade Learning Environment Breakout

In breakout you have four lives and the points for the bricks are valued differently. The reward that the agent receives is the points it receives after breaking a block.

### Task 1.1

Complete the `dqn.py` file. Complete the DQN Network and write the training loop. It is recommended to take a look at the paper for help, especially section 2 and 4.

### Task 1.2

Train by running `python3 dqn.py` and it will should record videos of your epochs so that you can see how your algorithm performs. Keep in mind this process might be quite slow and will utilize a GPU quite poorly.

### Extensions

Well done! Now feel free to try to use the DQN algorithm you have created on other games, keep in mind that it might be quite hard to learn some of them. Or you could read about extensions in the *Rainbow* paper.

## Questions

For any questions about this exercise feel free to post them on canvas or email me at filnil-8@student.ltu.se.

## References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.