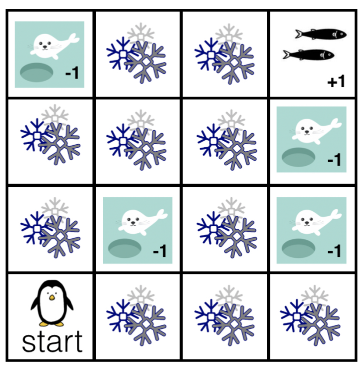
Deep Reinforcement Learning

Exercise 2: Sarsa, Q-learning

Lecturer: Armin Biess, Ph.D.

Due date: 20.7.2020

In this exercise you are asked to solve the frozen lake problem of exercise 1 with model-free RL algorithms. Given is the following MDP: $\gamma = 0.9$, reward $r(s, a, s') = -0.04$ except for the terminal states, where it is +1 and -1, respectively.



1. Sarsa [50 pts]

Implement SARSA on the girdworld with discount factor $\gamma = 0.9$. Find a suitable step-size parameter α and $\{\varepsilon_i\}$ decay-rate for GLIE. Add to the `env` a new method called `env.sarsa`. Write a plot function `plot_actionValues` that takes as input a state-action matrix $Q(s, a)$ and plots the optimal action values in the gridworld.

2. Q-Learning [50 pts]

Implement a Q-learning algorithm on the girdworld with discount factor $\gamma = 0.9$. Find a suitable step-size parameter α and $\{\varepsilon_i\}$ decay-rate for GLIE. Add to the `env` a new method called `env.Qlearning`.

The following functions are provided:

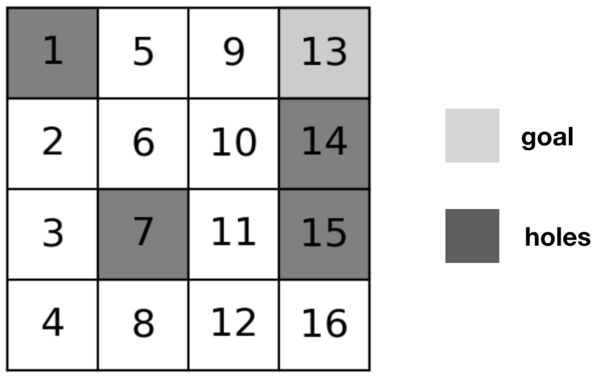
A `GridWorld` class with the following methods:

- `render` draws the gridworld and the agent in the gridworld
- `show` displays the gridworld and the agent in the gridworld
- `reset` sets the agent back to the starting state. It can take a binary argument *exploring_start*, for which the agent starts at a random initial state (`true`) or at the initial state 4 (`false`). If no argument is given *exploring_start* is set `false`.

- `step` takes as input an *action* and returns a *next_observation*, *reward* and a binary variable *done* indicating whether the episode ended.
- `close` closes the gridworld
- `plot_stateValues` takes as input a vector of values $V(s)$ and plots the values in the gridworld
- `plot_policy` takes as input a vector $\pi(s)$ or matrix $\pi(a|s)$ and plots the policy

Remarks:

- Label the states as follows:



- Label the actions as follows: $\{N, E, S, W\} = \{1, 2, 3, 4\}$.
- Experiment with the functions by executing main.py:

```python
from World import World
import numpy as np

if __name__ == "__main__":

    env = World()
    env.reset()
    done = False
    t = 0
    env.show()
    while not done:
        env.render()
        action = np.random.randint(1, env.nActions + 1)
        next_state, reward, done = env.step(action)  # take a random action
        env.close()
        t += 1
        if done:
            print("Episode finished after {} timesteps".format(t + 1))
            break

env.close();
```