Exercise 3

Gymnasium and TD-Learning

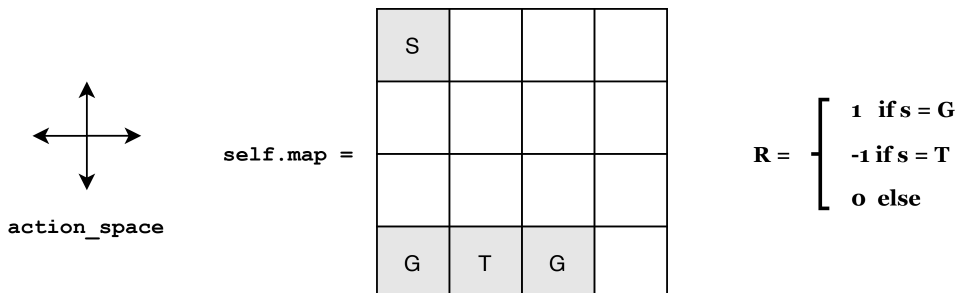
1 Gymnasium Gridworld

The Gymnasium/OpenAI Gym interface is a common standard for environments in reinforcement learning. It defines five methods that your environment has to implement. Providing the interface makes your environment compatible with nearly every RL framework (stable-baselines3, Tianshou, RLlib, ...).

The documentation and further information can be found here:

- `https://farama.org/Announcing-The-Farama-Foundation`
- `https://gymnasium.farama.org/`
- `https://github.com/Farama-Foundation/Gymnasium`

In this exercise, you will implement an environment based on the MDP definition. A simple grid world, similar to the last exercise, but with two different types of terminal states (two positives, **G**, and one negative **T**):



The agent starts in the cell marked **S**. At each time step, the agent can step up, right, down, or left if the agent reaches the goal (**G**), he receives a reward of 1, and the episode ends. If he falls into a trap (**T**), the episode ends with a reward of -1. Actions that would result in the agent leaving the board have no effect.

Programming Tasks:

1. `__init__()`. A `gymnasium` environment has to define its action and state space via two members, `action_space` and `observation_space`. Define the appropriate spaces inside the constructor. Also, add a member for the current environment state and assign the start state to it.

2. `reset()`. To reset the environment, a `reset()` method is required. This method should reset the environment state to a possible start state and return the observation for this state.

3. `step(action)`. To simulate a one-time step, the method `step(action)` is called with the agent's action. This method should simulate the environment dynamics and the agent's actions in one time step.

   `step(action)` should simulate one step in the environment and return a tuple `(observation, reward, terminated, truncated, info)` that describes the transition. `observation` is the agent's perception of the new state. `reward` is the agent's reward in the new state. `terminal` is `True` if the next state is terminal, `False` otherwise. `truncated` can be neglegted for exercise an can be set to `False`. `info` is a list or dictionary that contains additional (debug) information about the environment – for this implementation, you can return an empty dictionary ({}).

4. `render().` This helper method visualizes the current state in a human- or agent-readable way. In our case, this method should print an ASCII representation of the current map and agent.

5. `close().` This method frees up any resources (file handles, sockets, ...) that your environment needs to operate. In our case, the provided implementation (`pass`) does not need to be adapted.

Before continuing to the second exercise, we encourage you to test your implementation to find potential errors and problems. You can run the `env_test.py` file to execute random actions in your environment.

2 Temporal Difference Learning

TD-Learning is a technique that allows us to solve MDPs without access to the state transitions P. Your task is implementing a TD-Learning agent that solves the environment you implemented in Exercise 1. The file `td_agent.py` contains the skeleton code for this exercise. We supply testing code that deploys a random policy and visualizes the value function estimated through TD-Learning implementation.

Programming Tasks

1. `learn(n_timesteps).` This method currently implements `n_timesteps` of environment interaction via selecting a random action and deploying it in the environment. Implement TD-Learning and update the array `self.V` holding the current approximation at every time step. Run the script and examine the V estimates.

2. **Policy** `action(s).` Currently, this method implements a random policy. Implement a better policy (e.g., by mapping states to actions with a dictionary) and observe how the TD estimate of V changes visually.